

# **CPS**

---

DataScience con Python

# Salve!

---

- Massimo Regoli
- regoli@uniroma2.it
  - [CPS2023] nel subject
- Studio:
  - Facoltà di Ingegneria,
  - Edificio Informazione,
  - Piano terra,
  - Stanza AT05
- Ricevimento:
  - Martedì su appuntamento
  - SOLO ONLINE

# Il corso

---

- Lezioni
  - Mercoledì 11:30–13:15, aula
- Più che altro esercitazioni su alcuni interessanti dataset
- Il corso si baserà sull'ambiente *Jupyter Lab* nella configurazione presente nel pacchetto *Anaconda*
  - Ma nulla vieta di usare altre piattaforme
- Non verranno registrate le esercitazioni né fornito lo *streaming online*

# L'esame

---

- Come avrete sentito dal prof. Monte, l'esame sarà diviso in scritto ed orale
  - Nella prova orale sarà richiesta la presentazione di un progetto di corso
  - Progetto che deve essere richiesto da un gruppo (\*) attraverso Teams
  - Il progetto verrà consegnato il giorno dell'esame
    - È cosa buona e giusta che tutti i membri del gruppo facciano l'esame durante lo stesso appello
    - La discussione del progetto potrà avvalersi di un file di presentazione
    - Il contenuto della consegna dovrà comprendere:
      - Un manuale dei requirements e di uso dell'applicazione
      - Un file in python (notebook o altro) contenente tutto il codice
      - Tutti i file necessari per il funzionamento del progetto
  - Le richieste di progetto inizieranno (\*) da un gruppo un insieme di almeno 2 studenti verso dicembre

# Domande

- Ci sono dei dubbi?

- Q/A:

- ● ● ● ● ● ● ● ● ●

# Anaconda

---

# Data science technology for a better world.

Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.

[Download !\[\]\(eafc244b53721dd1ec133f0772f70fc7\_img.jpg\)](#)

For Windows

Python 3.9 • 64-Bit Graphical Installer • 594 MB

[Get Additional Installers](#)



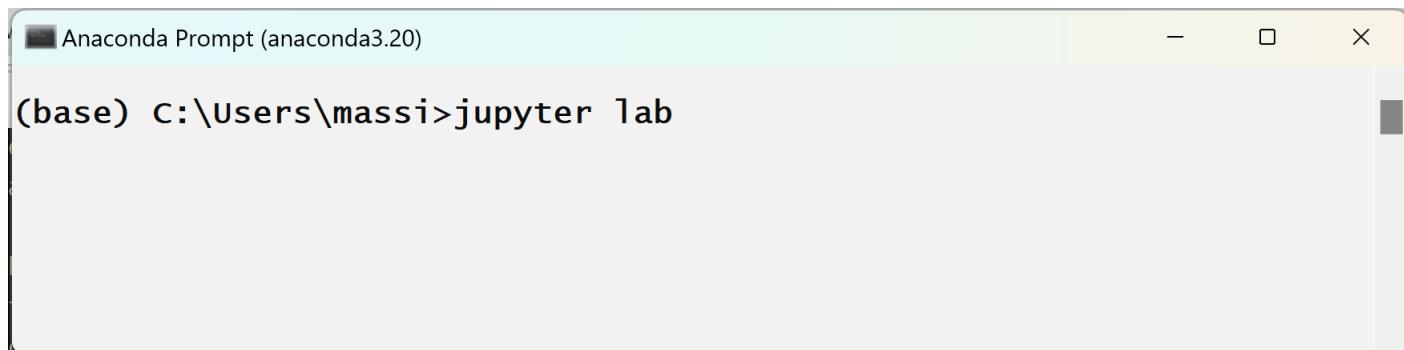
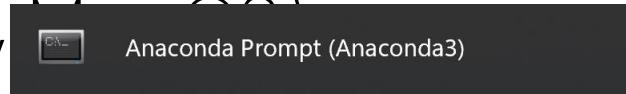
# Jupyter Notebook

---

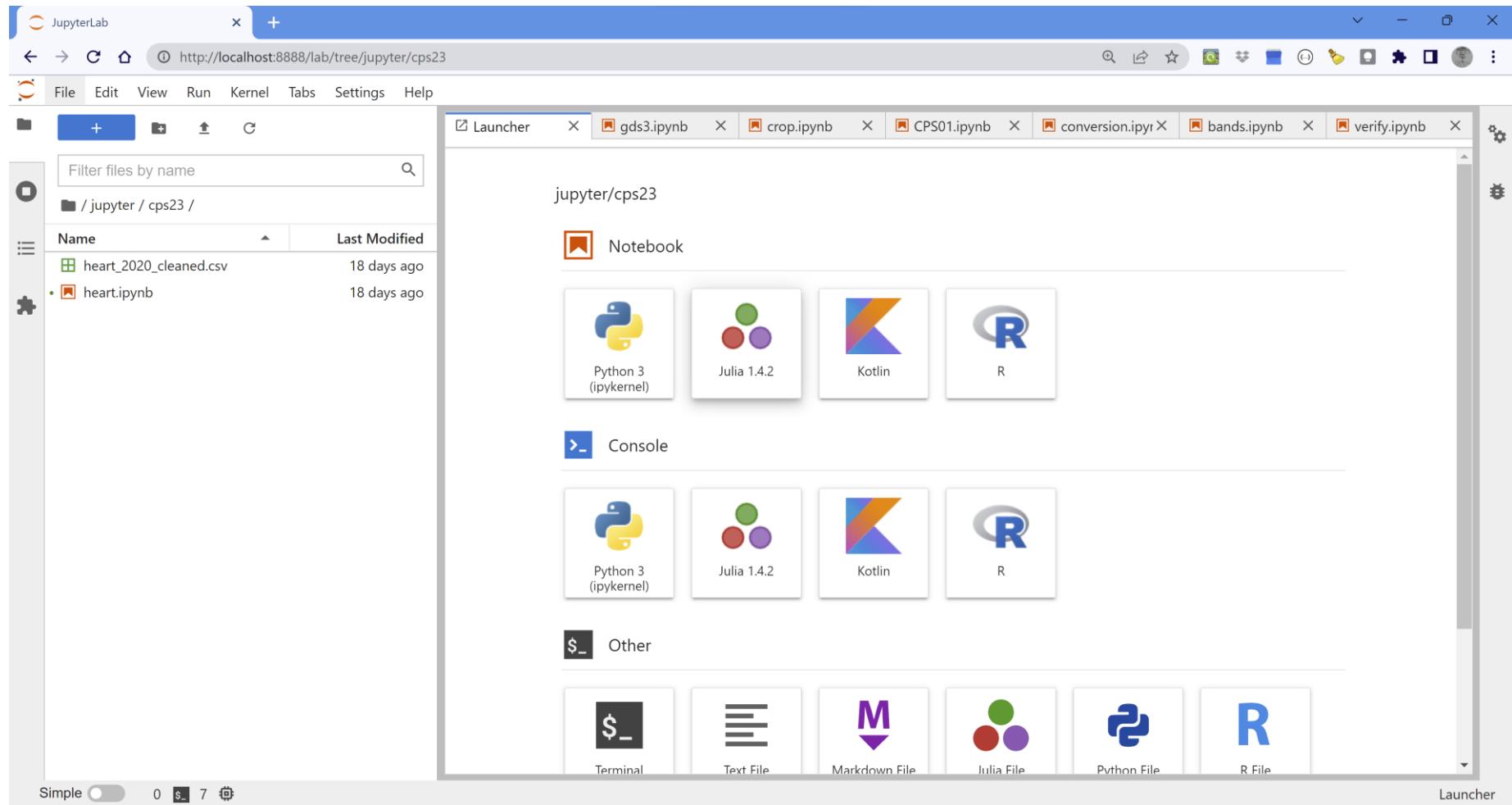
- *Jupyter Notebook* è un'applicazione Web open source che permette di creare e condividere documenti testuali interattivi, contenenti oggetti quali equazioni, grafici e codice sorgente eseguibile.
- *Jupyter* è diventato uno standard *de-facto* per data scientist perché offre la possibilità di realizzare, documentare e condividere analisi di dati all'interno di un framework [cit. ]

# Best practice

- Io preferisco *jupyter lab*
  - Interfaccia migliore e moderna
  - Pluginable
- Lanciare *jupyter lab* tramite Anaconda Navigator può risultare un processo lungo e farraginoso, meglio usare il prompt dei comandi (sia Windows, Linux, Mac OS)



# Jupyter lab



# Notebook - 1

The screenshot shows the Jupyter Lab interface. On the left, there is a file browser window titled "Untitled.ipynb (8) - JupyterLab". The address bar indicates the URL is <http://localhost:8888/lab/tree/jupyter/cps23/CPS01.ipynb#Esercitazione-1>. The file browser lists files in the directory /jupyter / cps23 /, including "CPS01.ipynb", "heart\_2020\_cleaned.csv", and "heart.ipynb". The main workspace contains an open notebook titled "Esercitazione 1" and "Notebook di Jupyter Lab". The notebook content includes:

## Esercitazione 1

### Notebook di Jupyter Lab

Un notebook di jupyter lab permette di alternare codice python a notazioni in formato markdown.

Il formato markdown permette di inserire delle annotazioni formattate testualmente co effetti come:

1. *corsivo*
2. **grassetto**
3. ***corsivo grassetto***
4. `testo verbatim per codice`
5. Stili di
  - A. Titolo (un #)
  - B. Sottotitolo (due ##)
  - C. ecc.
6. Elenchi numerati
  - elenchi
  - puntati
  - tutti e due anche in
    - forma nidificata
      - come qui

# Notebook - 2

The screenshot shows a Jupyter Notebook interface with the following components:

- Header:** CPS01.ipynb - JupyterLab, http://localhost:8888/lab/tree/jupyter/cps23/CPS01.ipynb
- File Menu:** File, Edit, View, Run, Kernel, Tabs, Settings, Help
- Toolbar:** +, -, ↑, ↓, C, G
- File Explorer:** Filter files by name, showing /jupyter / cps23 / with files CPS01.ipynb (seconds ago), heart\_2020... (18 days ago), and heart.ipynb (18 days ago).
- Code Cell:** [2]:

```
import random
x = [1, 2, 3, 4, 5]
random.shuffle(x)
```
- Output Cell:** [3]:

```
x
[5, 3, 4, 2, 1]
```
- Console:** Python 3 (ipykernel)

```
Python 3.7.11 (default, Jul 27 2021, 09:42:29) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced
Interactive Python. Type '?' for help.
```
- Status Bar:** Simple, 0 \$ 2, Python 3 (ipykernel) | Idle, Mode: Command, Ln 1, Col 1, CPS01.ipynb

# Alzate la mano

---

1. Chi conosce Python?
2. Chi conosce Jupyter Lab
3. Sapete cos'è un codice in formato Markdown?
4. Chi conosce Numpy e/o Pandas?

Per conoscenza intendo che siate capaci di scrivere un qualcosa complesso in quel linguaggio o libreria

# Librerie

---

- Librerie di base per Data Science
  - math
    - Raccolta di funzioni matematiche
  - random
    - Generatori di numeri casuali
  - datetime
    - Gestione delle date
  - time
    - Gestione del tempo
  - json
    - Raccolta di funzioni per la conversione dict  $\Leftrightarrow$  json

# Librerie

---

- Librerie avanzate per Data Science
  - numPy, sciPy
    - Calcolo vettoriale (alla Matlab)
    - numpy.random (best random)
  - pandas
    - Analisi dati
  - matplotlib
    - Grafici
  - seaborn
    - Grafici
  - sqlite3, dataset, mysql.connector
    - database

# Librerie

---

- Librerie specifiche per Data Science
  - *BeautifulSoup*
    - Scraping Web
  - *Selenium*
    - Web Browser automation

# Librerie

---

- Altre utili librerie
  - requests
    - Accesso a internet

# **NUMPY**

---

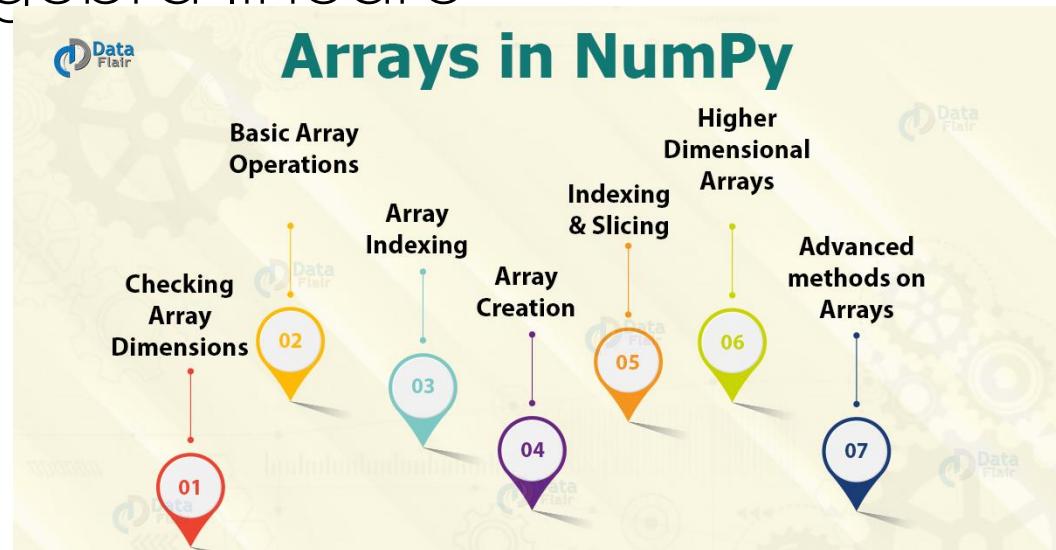
# Introduzione

---

- numpy è una libreria Python che aggiunge un importante supporto a *grandi matrici e array multidimensionali*
- Introduce una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati
- Molto utile nell'ambito della *data science*
- Alla base di molte altre librerie che useremo in questo corso

# Caratteristiche

- Alte prestazioni
- Codice base C/C++
- Gestione dati multidimensionali
- Funzioni broadcasting
- Funzione base di algebra lineare



# Installazione

Seleziona Anaconda Powershell Prompt (anaconda3.20)

```
(base) PS C:\Users\massi> conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

# Importazione

---

```
[1]: import numpy as np
```

```
[2]: np.version.full_version
```

```
[2]: '1.20.3'
```

```
[3]: np.version.version
```

```
[3]: '1.20.3'
```

# Array in numpy

---

- numpy ha una speciale classe denominata `numpy.ndarray` che assomiglia alla classe `list` di python
- Ma:
  - ammette solo tipi omogenei (ma strutturati)
  - è orientata al calcolo numerico vettoriale
- Per creare un oggetto di classe `numpy.ndarray` dobbiamo usare il relativo costruttore:

```
numpy.array(object,  
            dtype=None,  
            *,  
            copy=True,  
            order='K',  
            subok=False,  
            ndmin=0,  
            like=None)
```

# Prima

---

```
file_wh = open("weight-height.csv") # apro il file
sex = [] # inizializzo 4 liste
weight = []
height = []
bmi = []
lb_to_kg = 0.453592 # costanti di conversione
in_to_mt = 0.0254

file_wh.readline() # skip prima riga (header)
for line in file_wh:
    fields = line.split(",") # separo la riga
    sex.append(fields[0].strip('')) # aggiungo il record
    height.append(float(fields[1]) * in_to_mt)
    weight.append(float(fields[2]) * lb_to_kg)
    bmi.append(weight[-1] / height[-1]**2) # calcolo il bmi
file_wh.close()
```

# Adesso

---

```
import numpy as np

dataset = np.genfromtxt('weight-height.csv',
                       dtype=( 'U10', 'f', 'f'),
                       skip_header=1,
                       delimiter=",",
                       converters={0: lambda s:s[1:-1]},
                       names=["Sesso", "Altezza", "Peso"])
bmi = dataset["Peso"] * lb_to_kg / (dataset["Altezza"] * in_to_mt)**2
```

# genfromtxt

---

## numpy.genfromtxt

```
numpy.genfromtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None,
skip_header=0, skip_footer=0, converters=None, missing_values=None, filling_values=None,
usecols=None, names=None, excludelist=None, deletechars=" !#$%&'()*+,
-./:;=>?@[\\]^{|}~, replace_space='_', autostrip=False, case_sensitive=True,
defaultfmt='f%i', unpack=None, usemask=False, loose=True, invalid_raise=True,
max_rows=None, encoding='bytes', *, ndmin=0, Like=None) #
```

[\[source\]](#)

# Analisi

---

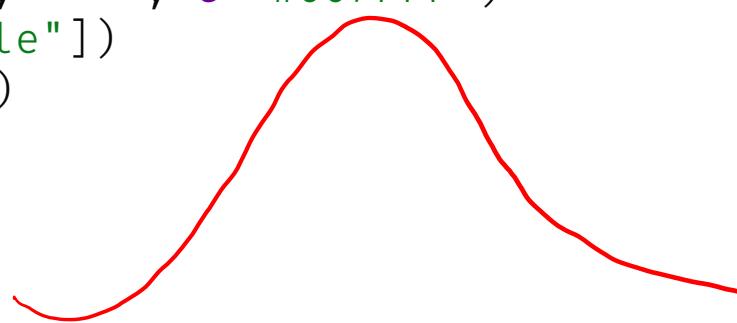
```
dataset = np.genfromtxt('weight-height.csv',  
    dtype descrive il  
    record del file  
    skip_header salta  
    la prima colonna  
    (header)  
delimiter descrive  
    il separatore dei  
    record  
converter è una  
    lambda da che  
    viene applicata ad  
    ogni record letto  
    names associa i  
    nomi degli  
    attributi  
    dtype=( 'U10' , 'f' , 'f' ),  
    skip_header=1,  
    delimiter="," ,  
    converters={0: lambda s:s[1:-1]},  
    names=[ "Sesso" , "Altezza" , "Peso" ])
```

# Interazione con matplotlib

- La classe `numpy.ndarray` è pienamente supportata da *matplotlib*

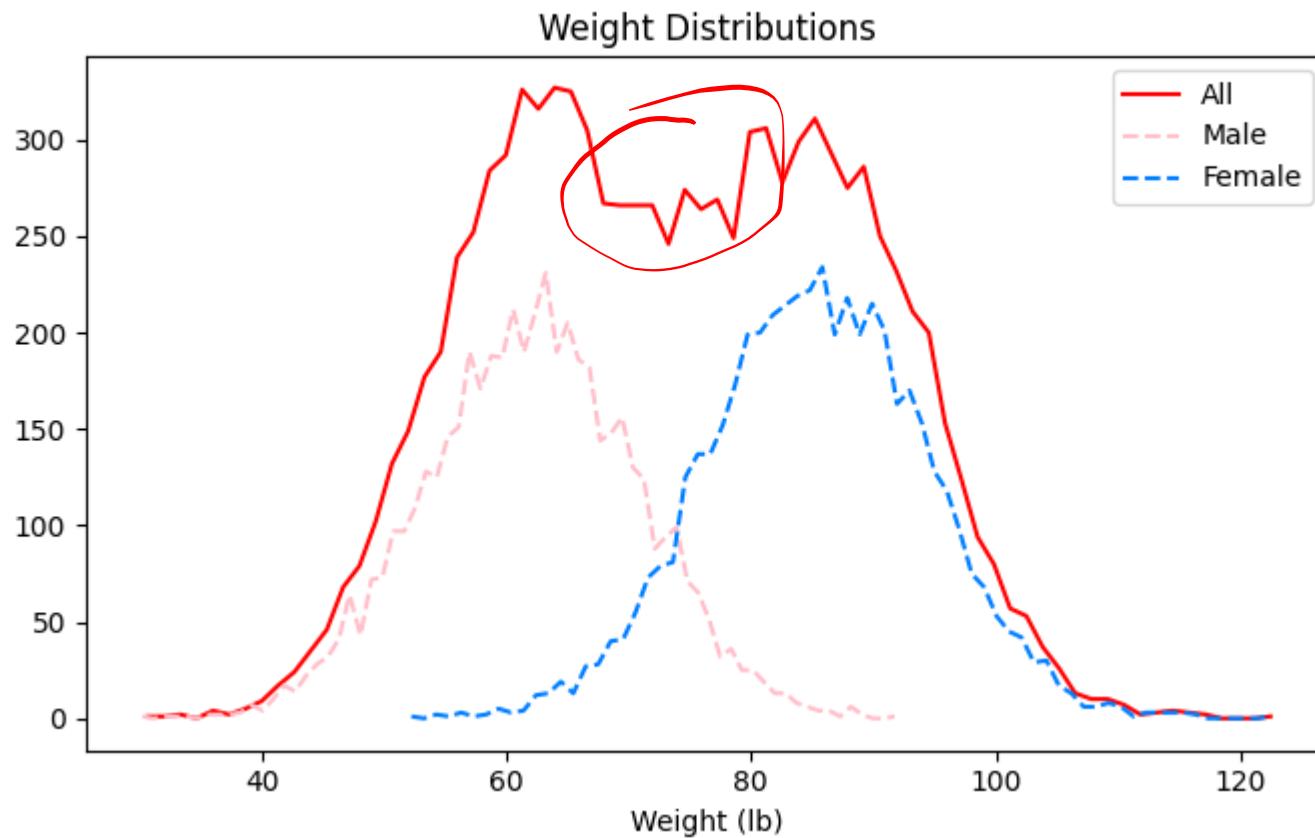
```
import matplotlib.pyplot as plt

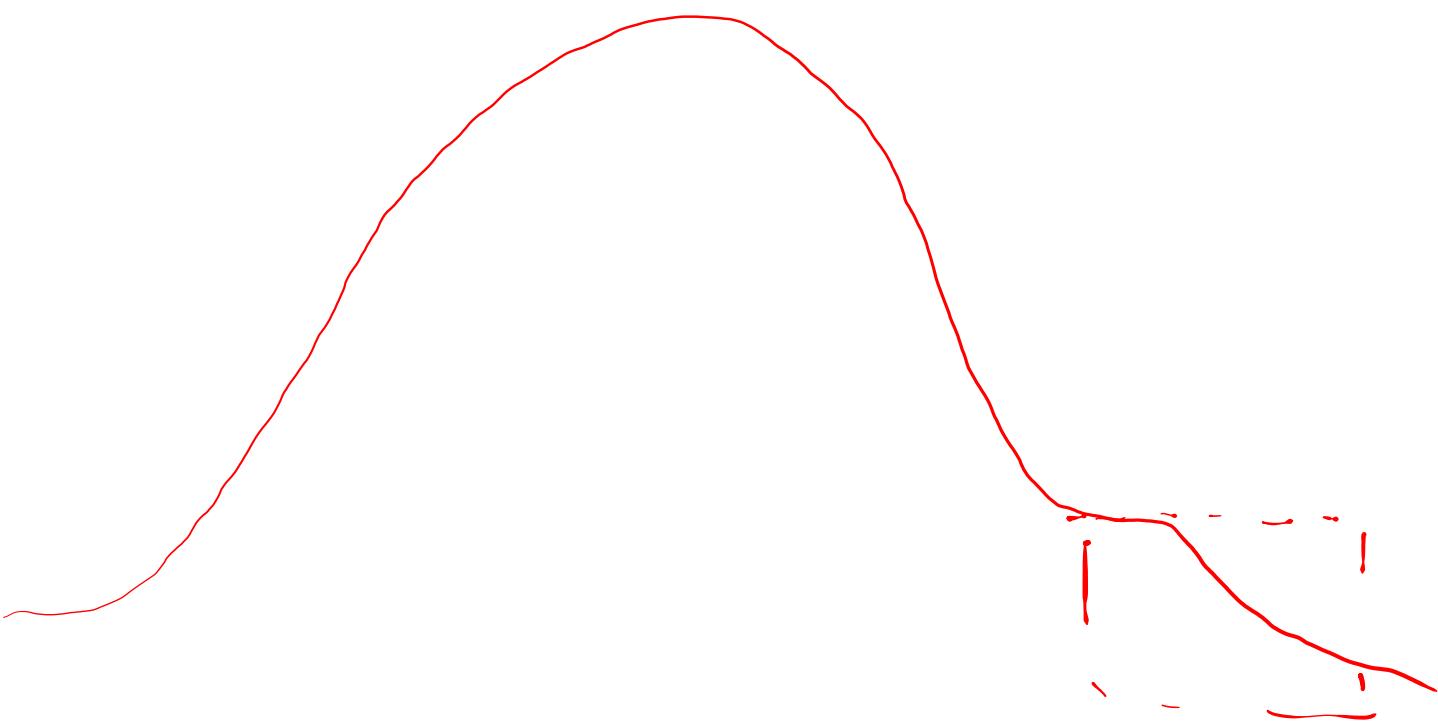
(values, intervals) = np.histogram(np_weight, bins=70)
(fvalues, fintervals) = np.histogram(np_weight[5000:], bins=70)
(mvalues, mintervals) = np.histogram(np_weight[:5000], bins=70)
plt.figure(figsize=(8,4.5))
plt.plot(intervals[1:], values, 'r')
plt.plot(fintervals[1:], fvalues, '--', c='#FFC0CB')
plt.plot(mintervals[1:], mvalues, '--', c='#007FFF')
plt.legend(["All", "Male", "Female"])
plt.title("Weight Distributions")
plt.xlabel("Weight (lb)")
plt.show()
```



# Integrazione con matplotlib – 2

---





# Overload operatori

- Gli operatori aritmetici e booleani hanno per gli array *numpy* un comportamento diverso dalle liste:
  - La somma di due *liste* crea una *lista* che è la concatenazione dei due addendi
- Per *numpy* la somma, la sottrazione, la divisione e tutte le altre operazioni binarie aritmetiche hanno il seguente comportamento:

$$l^{op} = l^1 \oplus l^2 \mid l_i^{op} = l_i^1 \oplus l_i^2 \quad (\oplus = \text{operatore}) \quad \forall i$$

*broadcasting*

- La stessa cosa vale per gli operatori booleani restituendo, in questo caso un array di booleani
- Ovviamente gli array devono essere della stessa lunghezza.

# Esempio

---

```
l1 = np.array([7, 2, 6, 4, 10])
l2 = np.array([2, 4, 5, 8, 4])
print(l1 / l2) # [3.5 0.5 1.2 0.5 2.5]
print(l1 * l2) # [14 8 30 32 40]
print(l1 % l2) # [1 2 1 4 2]

print(l1 > l2) # [ True False True False True]
print(l1 < l2) # [False  True False True False]
```

# Selezione (Slicing)

- Le liste Python ammettono alcune forme di selezione (*slicing*) legate alla posizione degli elementi
  - Possiamo ad esempio usando l'operatore parentesi quadra ([ ]) selezionare un sottoinsieme lineare
    - [a:b:c] dove a è l'elemento iniziale, b il finale e c lo step
    - a, b, c sono opzionali
- La stessa cosa fa anche l'array di numpy
- Ma fa molto di più!
- Ammette come indice un

## Array Indexing



# Esempio

---

```
import numpy as np

long_array = np.arange(10)
bool_array = [True, False, True, False,
              True, False, True, False,
              True, False]

print(long_array[bool_array])
```

[0 2 4 6 8]

# Esempio

---

```
# standard slicing
print(bmi[0:10000:1000])

# overload operatore > e slicing:
oversize = bmi > 32 # creo array
print(bmi[oversize])
# in breve
print(bmi[bmi > 32])
```

# Metodi

---

- La classe *ndarray* implementa vari metodi che possiamo dividere in categorie:
  - Conversione di array
  - Manipolazioni di forma
  - Selezione e manipolazione di elementi
  - Calcolo
  - Operatori matematici
    - overloaded
    - functional
  - Costruttori
  - Funzioni statistiche, trigonometriche, esponenziali, ...

# Funzioni matematiche

---

- Come esempio di funzioni vediamo alcune implementazioni di funzioni matematiche che agiscono non su numeri ma su array (Matlab style!)

## Trigonometric functions

---

**sin**(x, /[, out, where, casting, order, ...]) Trigonometric sine, element-wise.

---

**cos**(x, /[, out, where, casting, order, ...]) Cosine element-wise.

---

**tan**(x, /[, out, where, casting, order, ...]) Compute tangent element-wise.

---

**arcsin**(x, /[, out, where, casting, order, ...]) Inverse sine, element-wise.

---

**arccos**(x, /[, out, where, casting, order, ...]) Trigonometric inverse cosine, element-wise.

---

**arctan**(x, /[, out, where, casting, order, ...]) Trigonometric inverse tangent, element-wise.

# Trattamento dati

## Sums, products, differences

---

**prod(a[, axis, dtype, out, keepdims, ...])** Return the product of array elements over a given axis.

---

**sum(a[, axis, dtype, out, keepdims, ...])** Sum of array elements over a given axis.

---

**nanprod(a[, axis, dtype, out, keepdims, ...])** Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.

---

**nansum(a[, axis, dtype, out, keepdims, ...])** Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.

---

**cumprod(a[, axis, dtype, out])** Return the cumulative product of elements along a given axis.

---

**cumsum(a[, axis, dtype, out])** Return the cumulative sum of the elements along a given axis.

---

**nancumprod(a[, axis, dtype, out])** Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.

---

**nancumsum(a[, axis, dtype, out])** Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.

# Trattamento dati

---

## Text files

---

**loadtxt**(fname[, dtype, comments, delimiter, ...]) Load data from a text file.

---

**savetxt**(fname, X[, fmt, delimiter, newline, ...]) Save an array to a text file.

---

**genfromtxt**(fname[, dtype, comments, ...]) Load data from a text file, with missing values handled as specified.

---

**fromregex**(file, regexp, dtype[, encoding]) Construct an array from a text file, using regular expression parsing.

---

**fromstring**(string[, dtype, count, like]) A new 1-D array initialized from text data in a string.

---

**ndarray.tofile**(fid[, sep, format]) Write array to a file as text or binary (default).

---

**ndarray.tolist()** Return the array as an `a.ndim`-levels deep nested list of Python scalars.

# Funzioni

---

- Come abbiamo visto, il concetto di *overload* degli operatori di base viene ulteriormente esteso alle funzioni
- numpy mette a disposizione tutta una serie di funzioni matematiche che agiscono sugli *array*.
- Queste funzioni introducono il concetto di *broadcast function*.
- Vediamo quali funzioni sono implementate:

# Funzioni: trigonometriche

[sin\(\)](#)

Trigonometric sine, element-wise.

[cos\(\)](#)

Cosine element-wise.

[tan\(\)](#)

Compute tangent element-wise.

[arcsin\(\)](#)

Inverse sine, element-wise.

[arccos\(\)](#)

Trigonometric inverse cosine, element-wise.

[arctan\(\)](#)

Trigonometric inverse tangent, element-wise.

[hypot\(x1, x2\)](#)

Given the "legs" of a right triangle, return its hypotenuse.

[arctan2\(x1, x2\)](#)

Element-wise arc tangent of  $x_1/x_2$  choosing the quadrant correctly.

[degrees\(\)](#)

Convert angles from radians to degrees.

[radians\(\)](#)

Convert angles from degrees to radians.

[deg2rad\(\)](#)

Convert angles from degrees to radians.

[rad2deg\(\)](#)

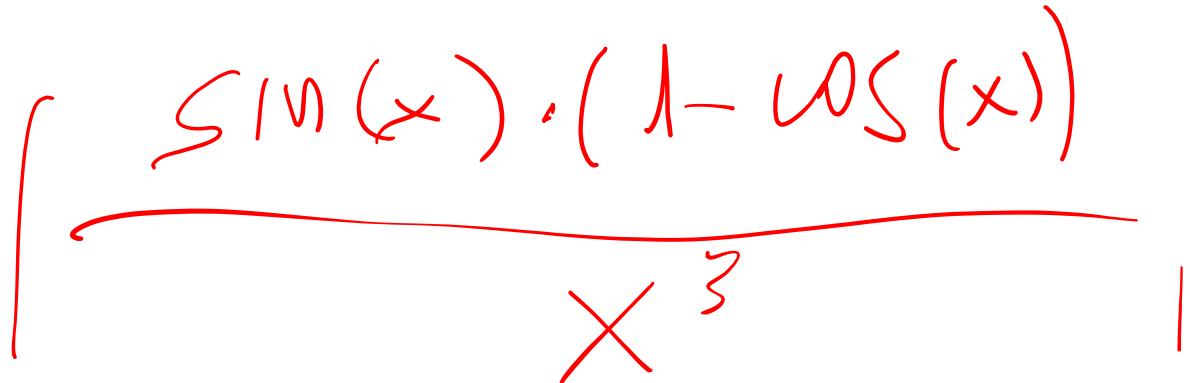
Convert angles from radians to degrees.

# Esempio

- Grazie a queste speciali funzioni che agiscono all'intero array e all'uso dell'overload degli operatori possiamo:

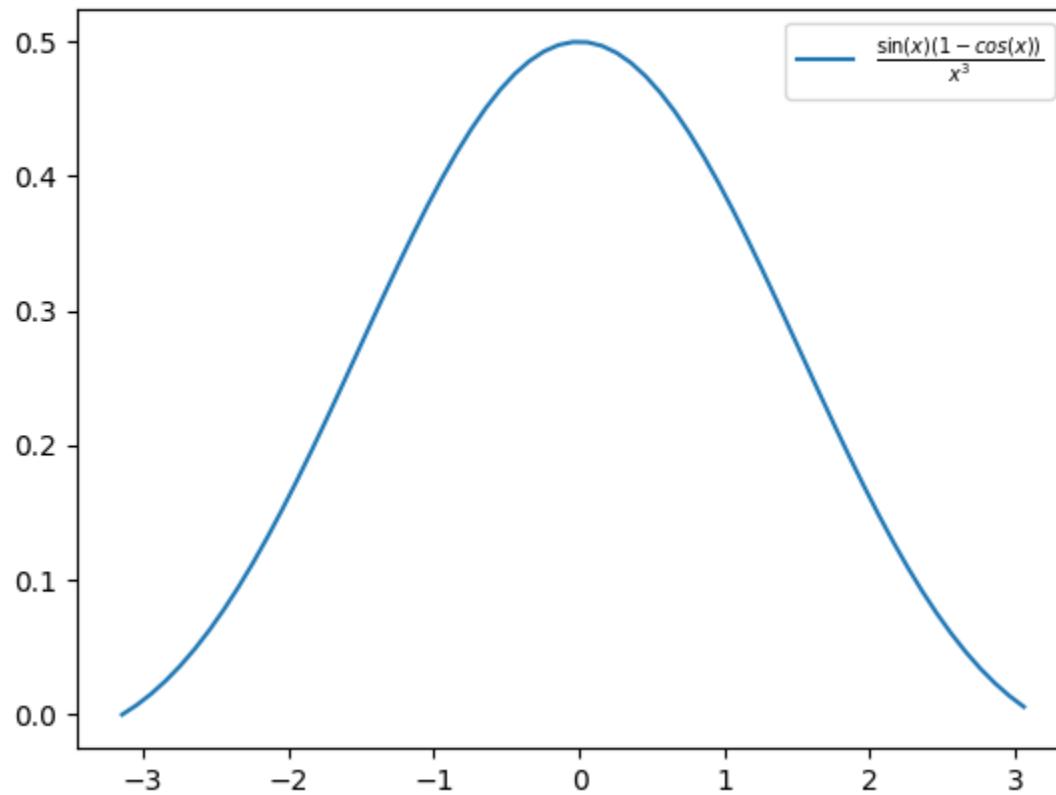
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi, np.pi, .1)
plt.plot(x, (np.sin(x) * (1-np.cos(x)))/(x**3))
plt.legend([r"\frac{\sin(x)(1-\cos(x))}{x^3}"])
plt.show()
```



# Esempio – cont.

---



# Funzioni: rounding

---

<u>around</u> (a[, decimals])	Evenly round to the given number of decimals.
<u>round_</u> (a[, decimals])	Round an array to the given number of decimals.
<u>rint</u> (x)	Round elements of the array to the nearest integer.
<u>fix</u> (x)	Round to nearest integer towards zero.
<u>floor</u> (x)	Return the floor of the input, element-wise.
<u>ceil</u> (x)	Return the ceiling of the input, element-wise.
<u>trunc</u> (x)	Return the truncated value of the input, element-wise.

# Funzioni: somma, prodotto, differenza

---

<a href="#"><u>prod</u></a> (a)	Return the product of array elements over a given axis.
<a href="#"><u>sum</u></a> (a)	Sum of array elements over a given axis.
<a href="#"><u>nanprod</u></a> (a)	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
<a href="#"><u>nansum</u></a> (a)	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
<a href="#"><u>cumprod</u></a> (a)	Return the cumulative product of elements along a given axis.
<a href="#"><u>cumsum</u></a> (a)	Return the cumulative sum of the elements along a given axis.
<a href="#"><u>nancumprod</u></a> (a)	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
<a href="#"><u>nancumsum</u></a> (a)	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
<a href="#"><u>diff</u></a> (a, ...)	Calculate the n-th discrete difference along the given axis.
<a href="#"><u>cross</u></a> (a, b, ...)	Return the cross product of two (arrays of) vectors.
<a href="#"><u>trapz</u></a> (y, ...)	Integrate along the given axis using the composite trapezoidal rule.

# Esempi

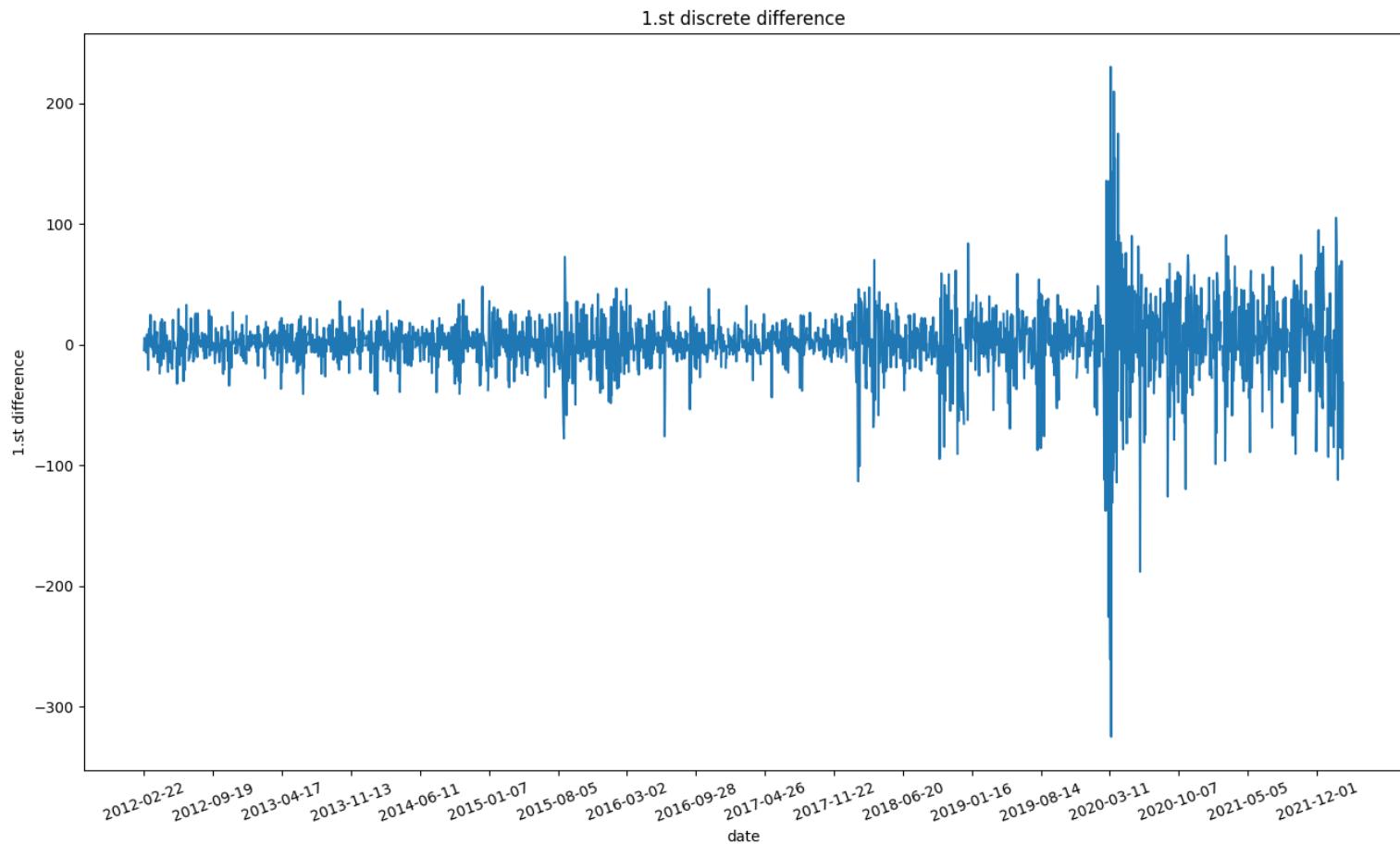
---

```
import numpy as np
import matplotlib.pyplot as plt

dt = np.dtype([('date', 'U10'), ('value', 'f8')])
sp500 = np.genfromtxt('sp500.csv',
                      delimiter=',',
                      dtype=dt,
                      skip_header=1,
                      missing_values={1: "."),
                      filling_values={1: np.nan},
                      names=["Date", "Value"])

plt.figure(figsize=(16, 9))
diff = np.diff(sp500["Value"], 1)
plt.plot(sp500["Date"][1:], diff)
plt.title(r"1.st discrete difference")
plt.xlabel("date")
plt.ylabel("1.st difference")
plt.xticks(range(0, len(sp500), 150), rotation=20)
plt.show()
```

# Esempio - 2



# Funzioni: logaritmiche/esponenziali

---

<u>exp</u> (x)	Calculate the exponential of all elements in the input array.
<u>expm1</u> (x)	Calculate $\exp(x) - 1$ for all elements in the array.
<u>exp2</u> (x)	Calculate $2^{**}p$ for all $p$ in the input array.
<u>log</u> (x)	Natural logarithm, element-wise.
<u>log10</u> (x)	Return the base 10 logarithm of the input array, element-wise.
<u>log2</u> (x)	Base-2 logarithm of x.
<u>log1p</u> (x)	Return the natural logarithm of one plus the input array, element-wise.
<u>logaddexp</u> (x1, x2)	Logarithm of the sum of exponentiations of the inputs.
<u>logaddexp2</u> (x1, x2)	Logarithm of the sum of exponentiations of the inputs in base-2.

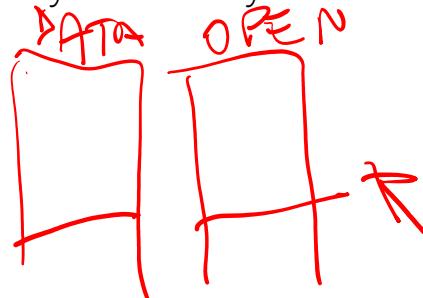
# Funzioni: estremi

---

<a href="#"><u>maximum</u></a> (x1, x2)	Element-wise maximum of array elements.
<a href="#"><u>fmax</u></a> (x1, x2)	Element-wise maximum of array elements.
<a href="#"><u>amax</u></a> (a)	Return the maximum of an array or maximum along an axis.
<a href="#"><u>nanmax</u></a> (a)	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
<a href="#"><u>minimum</u></a> (x1, x2)	Element-wise minimum of array elements.
<a href="#"><u>fmin</u></a> (x1, x2)	Element-wise minimum of array elements.
<a href="#"><u>amin</u></a> (a)	Return the minimum of an array or minimum along an axis.
<a href="#"><u>nanmin</u></a> (a)	Return minimum of an array or minimum along an axis, ignoring any NaNs.

# Funzioni: ricerca

<u>argmax</u> (a[, axis, keepdims])	Returns the indices of the maximum values along an axis.
<u>nanargmax</u> (a[, axis, keepdims])	Return the indices of the maximum values in the specified axis ignoring NaNs.
<u>argmin</u> (a[, axis, keepdims])	Returns the indices of the minimum values along an axis.
<u>nanargmin</u> (a[, axis, keepdims])	Return the indices of the minimum values in the specified axis ignoring NaNs.
<u>argwhere</u> (a)	Find the indices of array elements that are non-zero, grouped by element.
<u>nonzero</u> (a)	Return the indices of the elements that are non-zero.
<u>flatnonzero</u> (a)	Return indices that are non-zero in the flattened version of a.
<u>where</u> (condition, [x, y])	Return elements chosen from x or y depending on <i>condition</i> .
<u>searchsorted</u> (a, v[, side, sorter])	Find indices where elements should be inserted to maintain order.
<u>extract</u> (condition, arr)	Return the elements of an array that satisfy some condition.



# Esempi

---

```
vmax = np.nanmax(sp500["Value"])
vmin = np.nanmin(sp500["Value"])
indvmax = np.nanargmax(sp500["Value"])
indvmin = np.nanargmin(sp500["Value"])
dmax = sp500["Date"][indvmax]
dmin = sp500["Date"][indvmin]

print(f"Massimo storico {vmax} il {dmax}")
print(f"Minimo storico {vmin} il {dmin}")
```

Massimo storico 4796.56 il 2022-01-03  
Minimo storico 1278.04 il 2012-06-01

# Funzioni: Statistiche - 1

<u>median</u> (a[, axis])	Compute the median along the specified axis.
<u>average</u> (a[, axis, weights])	Compute the weighted average along the specified axis.
<u>mean</u> (a[, axis, dtype])	Compute the arithmetic mean along the specified axis.
<u>std</u> (a[, axis, dtype])	Compute the standard deviation along the specified axis.
<u>var</u> (a[, axis, dtype])	Compute the variance along the specified axis.
<u>nanmedian</u> (a[, axis])	Compute the median along the specified axis, while ignoring NaNs.
<u>nanmean</u> (a[, axis])	Compute the arithmetic mean along the specified axis, ignoring NaNs.
<u>nanstd</u> (a[, axis])	Compute the standard deviation along the specified axis, while ignoring NaNs.
<u>nanvar</u> (a[, axis])	Compute the variance along the specified axis, while ignoring NaNs.

1, 2, 3

$$\sum v_i \cdot w_i \rightarrow \sum w_i = 1$$
$$\sum v_i \cdot \frac{1}{n} = \frac{1}{n} \sum w_i \cdot \frac{\sum v_i}{n}$$

# Esempio

---

```
print(np.min(wh['height']),
      np.mean(wh['height']),
      np.max(wh['height']))
male = wh['gender'] == 'Male'
female = wh['gender'] == 'Female'

print(np.min(wh[male]['height']),
      np.mean(wh[male]['height']),
      np.max(wh[male]['height']))
print(np.min(wh[female]['height']),
      np.mean(wh[female]['height']),
      np.max(wh[female]['height']))
```

## Esempio – cont.

---

1.3782835864574663 1.6857360177724594 2.006568055598296  
1.4835353852664448 1.7532691860179221 2.006568055598296  
1.3782835864574663 1.6182028495269967 1.8640954809981702

# Funzioni: Statistiche - 2

---

[ptp](#)(a[, axis, out, keepdims])

Range of values (maximum - minimum) along an axis.

[percentile](#)(a, q[, axis])

Compute the q-th percentile of the data along the specified axis.

[nanpercentile](#)(a, q[, axis])

Compute the qth percentile of the data along the specified axis, while ignoring nan values.

[quantile](#)(a, q[, axis])

Compute the q-th quantile of the data along the specified axis.

[nanquantile](#)(a, q[, axis])

Compute the qth quantile of the data along the specified axis, while ignoring nan values.

---

[corrcoef](#)(x[, y, rowvar, bias, ddof, dtype])

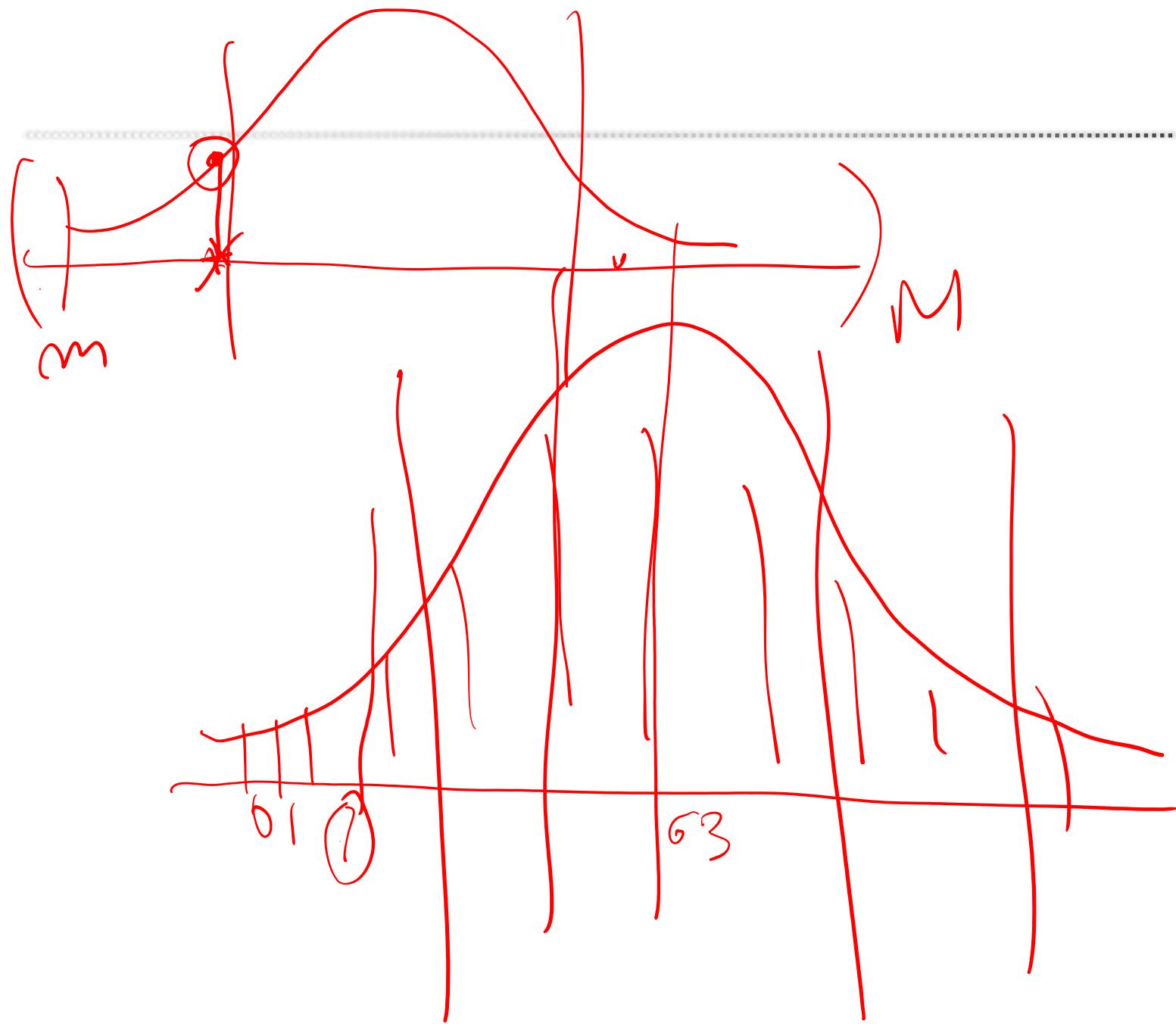
Return Pearson product-moment correlation coefficients.

[correlate](#)(a, v[, mode])

Cross-correlation of two 1-dimensional sequences.

[cov](#)(m[, y, rowvar, bias, ddof, fweights, ...])

Estimate a covariance matrix, given data and weights.



# Funzioni: Statistiche - 3

---

[histogram](#)(a[, bins, range, normed, weights, ...])

Compute the histogram of a dataset.

[histogram2d](#)(x, y[, bins, range, normed, ...])

Compute the bi-dimensional histogram of two data samples.

[histogramdd](#)(sample[, bins, range, normed, ...])

Compute the multidimensional histogram of some data.

[bincount](#)(x, /[, weights, minlength])

Count number of occurrences of each value in array of non-negative ints.

[histogram\\_bin\\_edges](#)(a[, bins, range, weights])

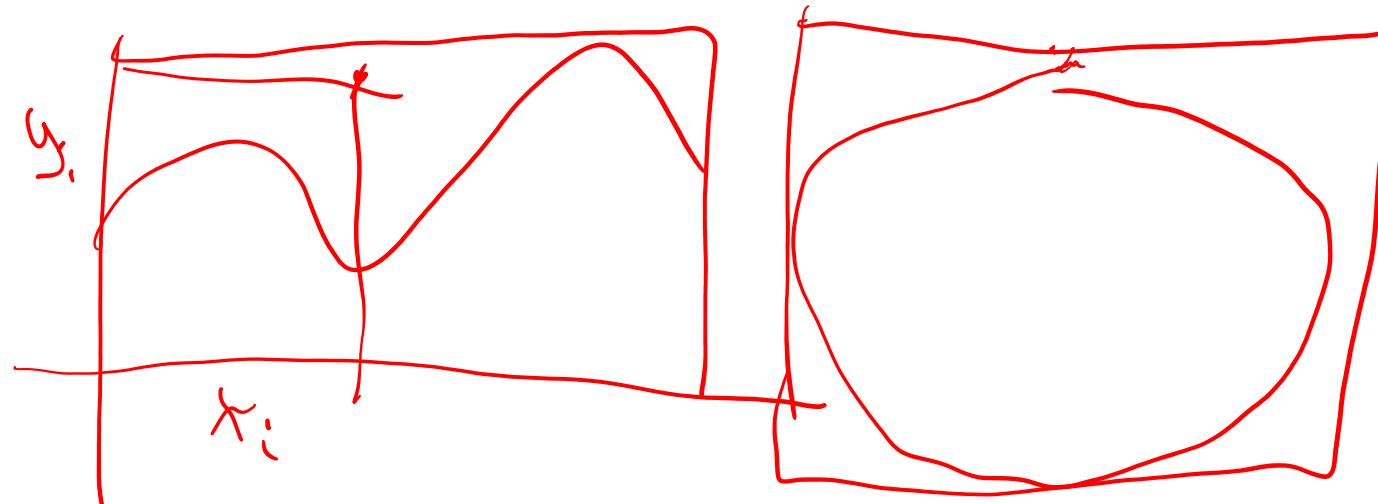
Function to calculate only the edges of the bins used by the [histogram](#) function.

[digitize](#)(x, bins[, right])

Return the indices of the bins to which each value in input array belongs.

# Funzioni: random

- random è un sotto modulo di numpy
- Per accedere, quindi, alle funzioni presenti nel modulo random di numpy bisogna preporre la stringa np. random
- Buona norma, inoltre, usare la classe Generator per creare una nuova istanza del generatore (ma non necessario)
- Esistono vari generatori di numeri casuali ma il *Mersenne Twister* è il generatore di default e per i nostri scopi va benissimo



# Funzioni: random

---

[integers](#)(low[, high, size, dtype, endpoint])

Return random integers from *low* (inclusive) to *high* (exclusive), or if *endpoint=True*, *low* (inclusive) to *high* (inclusive).

[random](#)([size, dtype, out])

Return random floats in the half-open interval [0.0, 1.0).

[choice](#)(a[, size, replace, p, axis, shuffle])

Generates a random sample from a given array

[bytes](#)(length)

Return random bytes.

[shuffle](#)(x[, axis])

Modify an array or sequence in-place by shuffling its contents.

[permutation](#)(x[, axis])

Randomly permute a sequence or return a permuted range.

[permuted](#)(x[, axis])

Randomly permute *x* along axis *axis*.

# Funzioni: random - 2

---

<a href="#"><u>beta</u>(a, b[, size])</a>	Draw samples from a Beta distribution.
<a href="#"><u>binomial</u>(n, p[, size])</a>	Draw samples from a binomial distribution.
<a href="#"><u>chisquare</u>(df[, size])</a>	Draw samples from a chi-square distribution.
<a href="#"><u>dirichlet</u>(alpha[, size])</a>	Draw samples from the Dirichlet distribution.
<a href="#"><u>exponential</u>([scale, size])</a>	Draw samples from an exponential distribution.
<a href="#"><u>f</u>(dfnum, dfden[, size])</a>	Draw samples from an F distribution.
<a href="#"><u>gamma</u>(shape[, scale, size])</a>	Draw samples from a Gamma distribution.
<a href="#"><u>geometric</u>(p[, size])</a>	Draw samples from the geometric distribution.
<a href="#"><u>gumbel</u>([loc, scale, size])</a>	Draw samples from a Gumbel distribution.
<a href="#"><u>hypergeometric</u>(ngood, nbad, nsample[, size])</a>	Draw samples from a Hypergeometric distribution.

# Funzioni: random - 3

---

[laplace](#)([loc, scale, size])

Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).

[logistic](#)([loc, scale, size])

Draw samples from a logistic distribution.

[lognormal](#)([mean, sigma, size])

Draw samples from a log-normal distribution.

[logseries](#)(p[, size])

Draw samples from a logarithmic series distribution.

[multinomial](#)(n, pvals[, size])

Draw samples from a multinomial distribution.

[multivariate\\_hypergeometric](#)(colors, nsample)

Generate variates from a multivariate hypergeometric distribution.

[multivariate\\_normal](#)(mean, cov[, size, ...])

Draw random samples from a multivariate normal distribution.

# Funzioni: random - 4

---

<u><a href="#">negative_binomial</a></u> (n, p[, size])	Draw samples from a negative binomial distribution.
<u><a href="#">noncentral_chisquare</a></u> (df, nonc[, size])	Draw samples from a noncentral chi-square distribution.
<u><a href="#">noncentral_f</a></u> (dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
<u><a href="#">normal</a></u> ([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.
<u><a href="#">pareto</a></u> (a[, size])	Draw samples from a Pareto II or Lomax distribution with specified shape.
<u><a href="#">poisson</a></u> ([lam, size])	Draw samples from a Poisson distribution.
<u><a href="#">power</a></u> (a[, size])	Draws samples in [0, 1] from a power distribution with positive exponent a - 1.

# Esempio

---

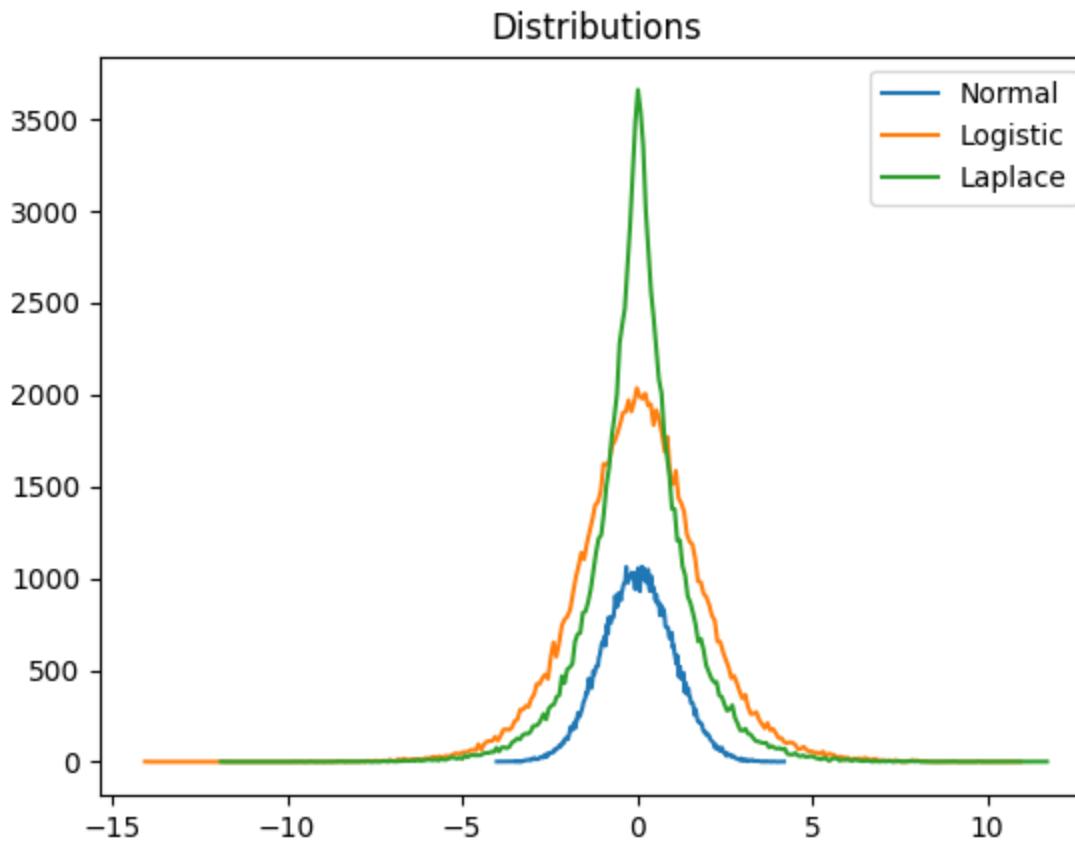
```
import numpy as np
import matplotlib.pyplot as plt

how_many = 100000
norm = np.random.normal(0, 1, how_many)
logi = np.random.logistic(0, 1, how_many)
lapl = np.random.laplace(0, 1, how_many)
hnorm, nint = np.histogram(norm, bins=int(np.sqrt(how_many)))
hlogi, hint = np.histogram(logi, bins=int(np.sqrt(how_many)))
hlapl, lapl = np.histogram(lapl, bins=int(np.sqrt(how_many)))

plt.plot(nint[1:], hnorm)
plt.plot(hint[1:], hlogi)
plt.plot(lapl[1:], hlapl)

plt.title("Distributions")
plt.legend(["Normal", "Logistic", "Laplace"])
plt.show()
```

# Esempio – cont.



# Funzioni: random - 5

<a href="#">rayleigh</a> ([scale, size])	Draw samples from a Rayleigh distribution.
<a href="#">standard_cauchy</a> ([size])	Draw samples from a standard Cauchy distribution with mode = 0.
<a href="#">standard_exponential</a> ([size])	Draw samples from the standard exponential distribution.
<a href="#">standard_gamma</a> (shape[, size])	Draw samples from a standard Gamma distribution.
<a href="#">standard_normal</a> ([size, dtype])	Draw samples from a standard Normal distribution (mean=0, stdev=1).
<a href="#">standard_t</a> (df[, size])	Draw samples from a standard Student's t distribution with $df$ degrees of freedom.
<a href="#">triangular</a> (left, mode, right[, size])	Draw samples from the triangular distribution over the interval [left, right].
<a href="#">uniform</a> ([low, high, size])	Draw samples from a uniform distribution.
<a href="#">vonmises</a> (mu, kappa[, size])	Draw samples from a von Mises distribution.
<a href="#">wald</a> (mean, scale[, size])	Draw samples from a Wald, or inverse Gaussian, distribution.
<a href="#">weibull</a> (a[, size])	Draw samples from a Weibull distribution.
<a href="#">zipf</a> (a[, size])	Draw samples from a Zipf distribution.

# Constructors - 1

---

- I costruttori sono metodi che permettono di creare array numpy
- Abbiamo già visto il costruttore principale: `array()`
- Altri costruttori:

[empty](#)(shape[, dtype, order, like])

Return a new array of given shape and type, without initializing entries.

[empty\\_like](#)(prototype[, dtype, order, subok, ...])

Return a new array with the same shape and type as a given array.

[eye](#)(N[, M, k, dtype, order, like])

Return a 2-D array with ones on the diagonal and zeros elsewhere.

[identity](#)(n[, dtype, like])

Return the identity array.

[ones](#)(shape[, dtype, order, like])

Return a new array of given shape and type, filled with ones.

[ones\\_like](#)(a[, dtype, order, subok, shape])

Return an array of ones with the same shape and type as a given array.

[zeros](#)(shape[, dtype, order, like])

Return a new array of given shape and type, filled with zeros.

[zeros\\_like](#)(a[, dtype, order, subok, shape])

Return an array of zeros with the same shape and type as a given array.

[full](#)(shape, fill\_value[, dtype, order, like])

Return a new array of given shape and type, filled with *fill\_value*.

[full\\_like](#)(a, fill\_value[, dtype, order, ...])

Return a full array with the same shape and type as a given array.

# Esempi

```
import numpy as np

empty_arr = np.empty((20), float)
print(empty_arr)

i33 = np.eye(3)
print(i33)

empty_like = np.empty_like(i33)
print(empty_like)

all_ones = np.ones(5)
print(all_ones)
all_zeros = np.zeros(5)
print(all_zeros)
fill_pi = np.full((3,3), np.pi)
print(fill_pi)
```

```
[0.  0.  0.3 1.  0.  0.  1.  1.  1.  1.]
[1.  1.  1.  0.  0.  1.  0.5 0.
 [0.  1. ]]
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
[1.  1.  1.  1.  1.]
[0.  0.  0.  0.  0.]
[[3.14159265 3.14159265 3.14159265]
 [3.14159265 3.14159265 3.14159265]
 [3.14159265 3.14159265 3.14159265]]
```

# Constructors - 2

<a href="#"><u>array</u>(object[, dtype, copy, order, subok, ...])</a>	Create an array.
<a href="#"><u>asarray</u>(a[, dtype, order, like])</a>	Convert the input to an array.
<a href="#"><u>asanyarray</u>(a[, dtype, order, like])</a>	Convert the input to an ndarray, but pass ndarray subclasses through.
<a href="#"><u>ascontiguousarray</u>(a[, dtype, like])</a>	Return a contiguous array (ndim $\geq 1$ ) in memory (C order).
<a href="#"><u>asmatrix</u>(data[, dtype])</a>	Interpret the input as a matrix.
<a href="#"><u>copy</u>(a[, order, subok])</a>	Return an array copy of the given object.
<a href="#"><u>frombuffer</u>(buffer[, dtype, count, offset, like])</a>	Interpret a buffer as a 1-dimensional array.
<a href="#"><u>fromfile</u>(file[, dtype, count, sep, offset, like])</a>	Construct an array from data in a text or binary file.
<a href="#"><u>fromfunction</u>(function, shape, *[, dtype, like])</a>	Construct an array by executing a function over each coordinate.
<a href="#"><u>fromiter</u>(iter, dtype[, count, like])</a>	Create a new 1-dimensional array from an iterable object.
<a href="#"><u>fromstring</u>(string[, dtype, count, like])</a>	A new 1-D array initialized from text data in a string.
<a href="#"><u>loadtxt</u>(fname[, dtype, comments, delimiter, ...])</a>	Load data from a text file.

# Esempio - genfromtxt

---

```
import numpy as np
dt = np.dtype([('sex', 'U10'), ('height', 'f8') , ('weight', 'f8')])
dataset = np.genfromtxt('weight-height.csv',
                        dtype=dt,
                        skip_header=1,
                        delimiter=",",
                        converters={0: lambda s:s[1:-1]},
                        names=["Sesso", "Altezza", "Peso"])
bmi = dataset["Peso"] * lb_to_kg / (dataset["Altezza"] * in_to_mt)**2
```

# Constructors - 3

---

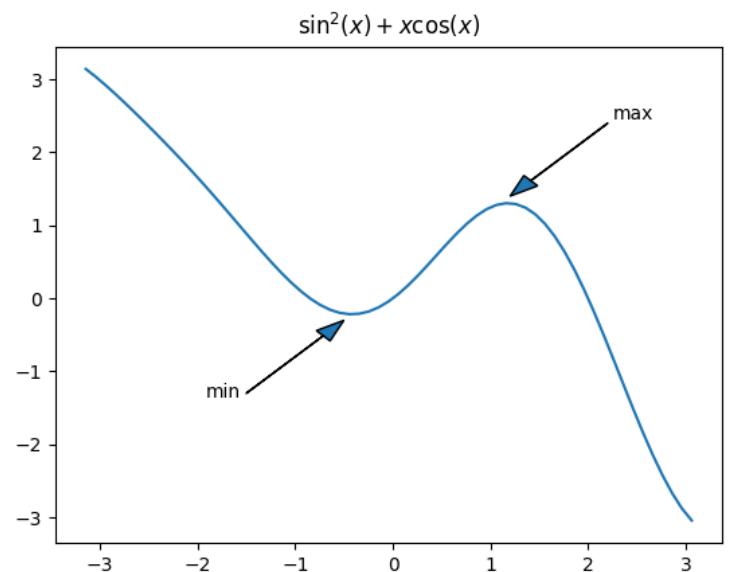
<a href="#"><u>arange</u></a> ([start,] stop[, step,][, dtype, like])	Return evenly spaced values within a given interval.
<a href="#"><u>linspace</u></a> (start, stop[, num, endpoint, ...])	Return evenly spaced numbers over a specified interval.
<a href="#"><u>logspace</u></a> (start, stop[, num, endpoint, base, ...])	Return numbers spaced evenly on a log scale.
<a href="#"><u>geomspace</u></a> (start, stop[, num, endpoint, ...])	Return numbers spaced evenly on a log scale (a geometric progression).

# Esempio

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi, np.pi, .1)
y = np.sin(x)*np.sin(x)+x*np.cos(x)

plt.plot(x, y)
plt.title(r"\sin^2(x)+x\cos(x)")
plt.arrow(2.2, 2.4, -1, -1,
          length_includes_head=True,
          head_width=0.2)
plt.text(2.25, 2.45, "max")
plt.arrow(-1.5, -1.3, 1, 1,
          length_includes_head=True,
          head_width=0.2)
plt.text(-1.55, -1.35, "min", ha='right')
plt.show()
```



# Esercizio 1 – Media Mobile

---

- numpy non ha una funzione che calcoli la *media mobile*
- Per fare questo costruiamone una usando le proprietà e gli operatori di numpy :
- Vi ricordo che la formula per il calcolo della media mobile su una serie storica  $\{y_t\}$  per  $t \in [1, T]$  su un periodo  $m \in \mathbb{N}^+$  è data da:

$$\bar{y}_t = \frac{1}{m} \sum_{i=0}^{m-1} y_{t-i} \quad \forall t \in [m, T]$$

# Esercizio 1 - Svolgimento

---

```
import numpy as np
import matplotlib.pyplot as plt

def moving_average(a, k):
    ret = [a[t-k:t].sum() for t in range(k, len(a))]
    return np.array(ret)/k

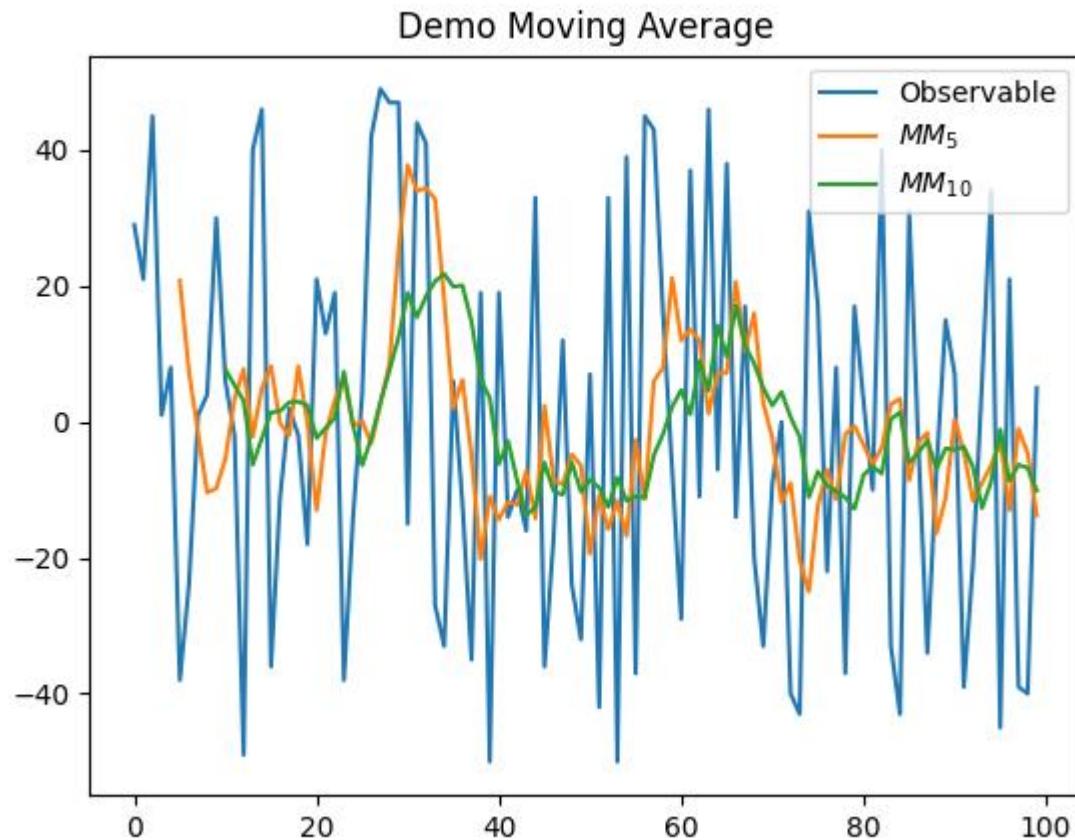
min_value = -50
max_value = 50
obs = 100
range1 = 5
range2 = 10

time_series = np.random.randint(min_value, max_value, obs)
mm1_time_series = moving_average(time_series, range1)
mm2_time_series = moving_average(time_series, range2)

plt.title("Demo Moving Average")
plt.plot(range(100), time_series)
plt.plot(range(range1, 100), mm1_time_series)
plt.plot(range(range2, 100), mm2_time_series)
plt.legend(["Observable", f"$MM_{range1}$", f"$MM_{{{{range2}}}}$"])
plt.show()
```

# Esercizio 1 - Risultati

---



# Materiale

---

- <https://numpy.org>
- <https://python.org>
- <https://matplotlib.org>
- <https://github.com/rougier/scientific-visualization-book>
- <https://data-flairing.training>

# pandas

---

# pandas

---

- *pandas* è una libreria Python contenente strutture e strumenti di dati di alto livello che sono stati creati per aiutare i programmatori Python a eseguire analisi sui dati.
- Lo scopo finale di *pandas* è di aiutare a scoprire rapidamente le informazioni nei dati, informazioni intese come significato sottostante.
- L'utilizzo di *pandas* nell'ambiente del *datascientist* Python è **fondamentale**
- *pandas* è già inclusa nell'ambiente Anaconda

# pandas

---

- Oggetti Series ed DataFrame veloci ed efficienti per la manipolazione dei dati con indicizzazione integrata
- Allineamento intelligente dei dati tramite indici ed etichette
- Gestione integrata dei dati mancanti
- Funzionalità per la conversione di dati disordinati in dati ordinati (riordino)
- Strumenti integrati per la lettura e la scrittura di dati tra strutture e file di dati in memoria, database e servizi Web
- Capacità di elaborare i dati memorizzati in molti formati comuni come CSV, Excel, HDF5 e JSON

# pandas

---

- Smart slicing basato su etichette, indicizzazione elaborata e sottoinsieme di set di dati di grandi dimensioni
- Le colonne possono essere inserite ed eliminate dalle strutture di dati per la mutabilità delle dimensioni
- Aggregazione o trasformazione di dati con una potente funzione di raggruppamento per eseguire divisioni di applicazione divisa su set di dati

# pandas

---

- Core
  - I due componenti principali di pandas sono **Series** e **DataFrame**.
  - **Series** è essenzialmente una colonna e un **DataFrame** è una tabella bidimensionale composta da una raccolta di **Series**

# Series

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0
6	Evan Turner	Boston Celtics	11.0

ser = pd.Series(df [ 'Name'])

ser = pd.Series(df [ 'Team'])

ser = pd.Series(df [ 'Number'])

DG

# Series

---

- Una `Series` pandas è un array etichettabile unidimensionale in grado di contenere dati di qualsiasi tipo (intero, stringa, float, oggetti Python, ecc.).
- Le etichette degli assi sono chiamate collettivamente *indici*.
- Una `Series` non è altro che una colonna in un foglio Excel.

# Costruttori

---

- I dati possono essere molte cose diverse:
  - un dict Python  $s = pd.Series(data, index=index)$
  - un ndarray
  - un valore scalare (come 5)
- L'indice passato è un elenco di etichette degli assi.
  - Pertanto, questo si differenzia in alcuni casi a seconda di quali dati sono:
    - ndarray
    - dict
    - valore scalare

# Series

- Creare una Series

```
In [1]: 1 import pandas as pd  
2 import numpy as np
```

```
In [2]: 1 array_np = np.array([1, 2, 3])  
2 ser = pd.Series(array_np)  
3 print(ser)
```

0 1  
1 2  
2 3  
dtype: int32

```
In [3]: 1 list_py = ['a', 1, True]  
2 ser = pd.Series(list_py)  
3 ser = ser.rename("NOME")  
4 print(ser)
```

0 a  
1 1  
2 True  
Name: NOME, dtype: object

```
In [4]: 1 dict = {'A' : 10,  
2           'B' : 20,  
3           'C' : 30}  
4  
5 ser = pd.Series(dict)  
6  
7 print(ser)
```

A 10  
B 20  
C 30  
dtype: int64

# Series

---

- Accesso
  - L'accesso ad un elemento della serie è possibile usare l'indice in qualunque forma

```
dict = {'A': 10,  
        'B': 20,  
        'C': 30}  
ser = pd.Series(dict)  
print(ser['A'])
```

```
list_py = ['a', 1, True]  
ser = pd.Series(list_py)  
print(ser[2])
```

# Series

---

- `Series.iloc[n]`
  - Ritorna la posizione relativa e non quella della etichetta
- `Series.loc[n]`
  - Ritorna la posizione indicizzata dall'etichetta
- `Series[n]`
  - Ritorna la posizione indicizzata dalla posizione assoluta

## .iloc[]

---

- .iloc[] è principalmente basato sulla posizione intera (da 0 a lunghezza-1 dell'asse), ma può anche essere utilizzato con un array booleano.
- Valore ammessi:
  - Intero
  - Una lista di interi
  - Un oggetto slice (es. 1:3)
  - Una array booleano
  - Una *callable function* con un argomento (la serie)

## .loc[ ]

---

- Accede a un gruppo di righe e colonne per etichetta o per una matrice booleana.
- .loc[ ] è principalmente basato su etichette, ma può anche essere utilizzato con un array booleano.
- Gli input consentiti sono:
  - Una etichetta singola
  - Una lista di etichette
  - Uno slice (es. 'a':'c')
  - Un array booleano
  - Una funzione *callable*

# .iloc[] vs .loc[]

	loc	iloc
A value	A single label or integer e.g. loc[A] or loc[1]	A single integer e.g. iloc[1]
A list	A list of labels e.g. loc[[A, B]]	A list of integers e.g. iloc[[1,2,3]]
Slicing	e.g. loc[A:B], A and B are included	e.g. iloc[n:m], n is included, m is excluded
Conditions	A bool Series or list	A bool list
Callable function	loc[lambda x: x[2]]	iloc[lambda x: x[2]]

# Series

---

```
list_py = [2, 3, 4, 5, 6, 7, 8]      4  6
ser = pd.Series(list_py)                5  7
ser2 = ser[4:]                         6  8
print(ser2)                           INDICE [5]
print("INDICE [5]")                   7
print(ser2[5])                        INDICE loc[5]
print("INDICE loc[5]")                 7
print(ser2.iloc[5])                   INDICE iloc[1]
print("INDICE iloc[5]")                7
print(ser2.iloc[1])                   1) INDICE [5:6]
print("1) INDICE [5:6]")              Series[], dtype: int64)
print(ser2[5:6])                      2) INDICE loc[5:6]
print("2) INDICE loc[5:6]")            5  7
print(ser2.loc[5:6])                  6  8
print("3) INDICE iloc[1:2]")          3) INDICE iloc[1:2]
print(ser2.iloc[1:2])                  5  7
```

# Series

- Operazioni su Series

```
In [39]: 1 data = pd.Series([5, 2, 3, 7], index=['a', 'b', 'c', 'd'])
2 data1 = pd.Series([1, 6, 4, 9], index=['a', 'b', 'd', 'e'])
```

Usando la versione funzionale è possibile descrivere cosa fare in caso di chiavi mancanti

```
In [40]: 1 data.add(data1, fill_value=0)
```

```
Out[40]: a    6.0
         b    8.0
         c    3.0
         d   11.0
         e    9.0
        dtype: float64
```

```
In [41]: 1 data.sub(data1, fill_value=0)
```

```
Out[41]: a    4.0
         b   -4.0
         c    3.0
         d    3.0
         e   -9.0
        dtype: float64
```

Usando la versione operazionale le chiavi mancanti producono un NaN

```
In [42]: 1 data-data1
```

```
Out[42]: a    4.0
         b   -4.0
         c    NaN
         d    3.0
         e    NaN
        dtype: float64
```

# Pandas vs Numpy

---

- Le Series pandas anche se a prima vista molto simili agli array di numpy sono in realtà assai diversi
- Ad esempio un Array numpy è un oggetto omogeneo mentre la Series pandas è eterogeneo:

# Pandas vs Numpy

---

```
import numpy as np
import pandas as pd

vett1 = [1, 1.4, True, 'ciao']

np_vett1 = np.array(vett1)
print(f"Tipo dati array numpy: {np_vett1.dtype}")
for i in range(len(np_vett1)):
    print(f"Dato originale {type(vett1[i])} " +
          f" tipo derivato numpy {type(np_vett1[i])}")

pd_vett1 = pd.Series(vett1)
print(f"Tipo dati array pandas: {pd_vett1.dtype}")

for i in range(len(pd_vett1)):
    print(f"Dato originale {type(vett1[i])} " +
          f" tipo derivato pandas {type(pd_vett1[i])}")
```

# Pandas vs Numpy

---

Tipo dati array numpy: <U32

Dato originale <class 'int'> tipo derivato numpy <class 'numpy.str\_'>

Dato originale <class 'float'> tipo derivato numpy <class 'numpy.str\_'>

Dato originale <class 'bool'> tipo derivato numpy <class 'numpy.str\_'>

Dato originale <class 'str'> tipo derivato numpy <class 'numpy.str\_'>

Tipo dati array pandas: object

Dato originale <class 'int'> tipo derivato pandas <class 'int'>

Dato originale <class 'float'> tipo derivato pandas <class 'float'>

Dato originale <class 'bool'> tipo derivato pandas <class 'bool'>

Dato originale <class 'str'> tipo derivato pandas <class 'str'>

# Series

- Conversioni

```
In [77]: 1 data = pd.Series([6, 5.2, '7', 7])
2 data1 = pd.to_numeric(data)
3 print("PRIMA", data.dtypes, "DOPO", data1.dtypes)
```

PRIMA object DOPO float64

```
In [78]: 1 data2 = data1.astype(int)
2 print("PRIMA", data1.dtypes, "DOPO", data2.dtypes)
```

PRIMA float64 DOPO int32

```
In [81]: 1 data3 = pd.Series(["2020-01-01", "2020-01-02", "2020-01-03", "NP"])
2 data4 = pd.to_datetime(data3, errors='coerce')
```

```
In [82]: 1 data4
```

```
Out[82]: 0    2020-01-01
1    2020-01-02
2    2020-01-03
3        NaT
dtype: datetime64[ns]
```

# Series – Operazioni Binarie

FUNCTION	DESCRIPTION
<u>add()</u>	Method is used to add series or list like objects with same length to the caller series
<u>sub()</u>	Method is used to subtract series or list like objects with same length from the caller series
<u>mul()</u>	Method is used to multiply series or list like objects with same length with the caller series
<u>div()</u>	Method is used to divide series or list like objects with same length by the caller series
<u>sum()</u>	Returns the sum of the values for the requested axis
<u>prod()</u>	Returns the product of the values for the requested axis
<u>mean()</u>	Returns the mean of the values for the requested axis
<u>pow()</u>	Method is used to put each element of passed series as exponential power of caller series and returned the results
<u>abs()</u>	Method is used to get the absolute numeric value of each element in Series/DataFrame
<u>cov()</u>	Method is used to find covariance of two series

# Esempio

---

```
import numpy as np
import pandas as pd

ser1 = pd.Series([1, 2, 3, 4, 5, 6],
                 index=['a', 'b', 'c', 'd', 'e', 'f'])
ser2 = pd.Series(np.random.randint(1, 7, 6),
                 index=np.random
                     .permutation(['a', 'b', 'c', 'd', 'e', 'f']))
ser3 = ser1 + ser2
print(ser3)
ser4 = pd.Series.add(ser2, ser1)
print(ser4)
ser5 = ser1.add(ser2)
print(ser5)
```

# Series – Metodi

FUNCTION	DESCRIPTION
<code>Series()</code>	A pandas Series can be created with the <code>Series()</code> constructor method. This constructor method accepts a variety of inputs
<code>combine_first()</code>	Method is used to combine two series into one
<code>count()</code>	Returns number of non-NA/null observations in the Series
<code>size</code>	Returns the number of elements in the underlying data
<code>name</code>	Method allows to give a name to a Series object, i.e. to the column
<code>is_unique</code>	Method returns boolean if values in the object are unique
<code>idxmax()</code>	Method to extract the index positions of the highest values in a Series
<code>idxmin()</code>	Method to extract the index positions of the lowest values in a Series
<code>sort_values()</code>	Method is called on a Series to sort the values in ascending or descending order
<code>sort_index()</code>	Method is called on a pandas Series to sort it by the index instead of its values
<code>head()</code>	Method is used to return a specified number of rows from the beginning of a Series. The method returns a brand new Series
<code>tail()</code>	Method is used to return a specified number of rows from the end of a Series. The method returns a brand new Series

# Esempio

---

```
print(f"ser4 è a valori unici? {ser4.is_unique}")  
ser4.name = "serie4"  
print(f"Il nome della serie adesso è: {ser4.name}")  
print(f"In ser4 ci sono {ser4.size} " +  
     f"elementi di cui {ser4.count()} non nulli")  
print(f"Il massimo in ser4 è " +  
     f"all'indice {ser4.idxmax()} e vale {ser4.max()}")
```

# Series – Metodi

FUNCTION	DESCRIPTION
<code>le()</code>	Used to compare every element of Caller series with passed series. It returns True for every element which is Less than or Equal to the element in passed series
<code>ne()</code>	Used to compare every element of Caller series with passed series. It returns True for every element which is Not Equal to the element in passed series
<code>ge()</code>	Used to compare every element of Caller series with passed series. It returns True for every element which is Greater than or Equal to the element in passed series
<code>eq()</code>	Used to compare every element of Caller series with passed series. It returns True for every element which is Equal to the element in passed series
<code>gt()</code>	Used to compare two series and return Boolean value for every respective element
<code>lt()</code>	Used to compare two series and return Boolean value for every respective element

# Esempi

```
► 1 11 = [0, 2, 3, 2, 1, 2, 3, 1, 2, 3, 0, 2]
  2 12 = [2, 3, 2, 1, 2, 3, 2, 0, 2, 0, 0, 1]
  3 s1 = pd.Series(11)
  4 s2 = pd.Series(12)
```

```
► 1 s1.ge(s2)
```

```
: 0    False
  1    False
  2     True
  3     True
  4    False
  5    False
  6     True
  7     True
  8     True
  9     True
 10    True
 11    True
dtype: bool
```

# Esempi

---

```
1 | s1[s1>=s2]
```

```
2      3
3      2
6      3
7      1
8      2
9      3
10     0
11     2
dtype: int64
```

# Series – Metodi

FUNCTION	DESCRIPTION
<code>clip()</code>	Used to clip value below and above to passed Least and Max value
<code>clip_lower()</code>	Used to clip values below a passed least value
<code>clip_upper()</code>	Used to clip values above a passed maximum value
<code>astype()</code>	Method is used to change data type of a series
<code>tolist()</code>	Method is used to convert a series to list
<code>get()</code>	Method is called on a Series to extract values from a Series. This is alternative syntax to the traditional bracket syntax
<code>unique()</code>	Pandas unique() is used to see the unique values in a particular column
<code>nunique()</code>	Pandas nunique() is used to get a count of unique values
<code>value_counts()</code>	Method to count the number of the times each unique value occurs in a Series
<code>factorize()</code>	Method helps to get the numeric representation of an array by identifying distinct values
<code>map()</code>	Method to tie together the values from one object to another
<code>between()</code>	Pandas between() method is used on series to check which values lie between first and second argument

# Esempio

---

```
import pandas as pd
import numpy as np

def pariedispari(k):
    if k % 2 == 0:
        return 'pari'
    else:
        return 'dispari'
```

```
s1 = pd.Series(
    np.random.randint(0, 10, 8))
for a, b in zip(s1, s1.clip(3,6)):
    print(f"{a} => {b}")

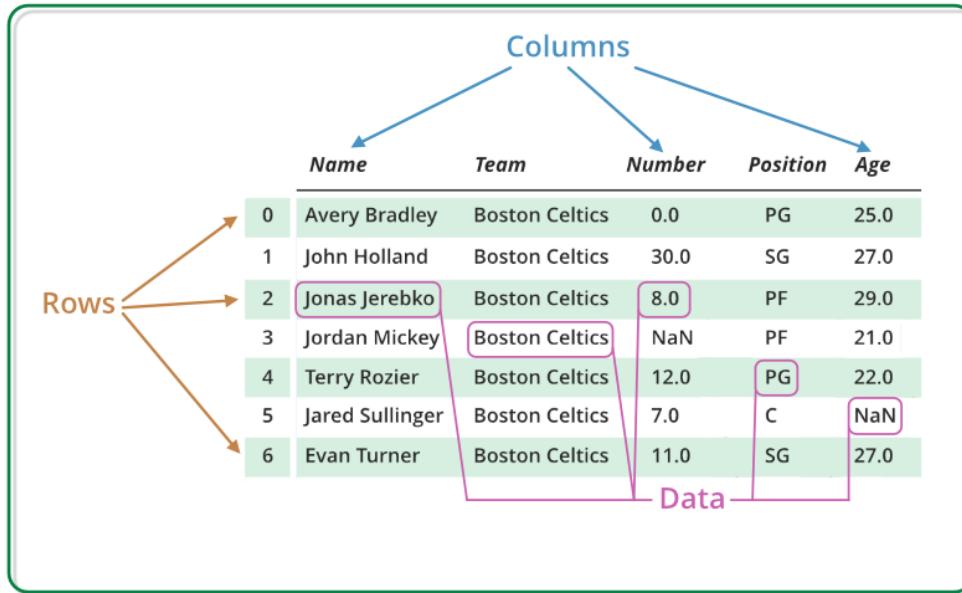
s2 = pd.Series([
    1, 2.4, False, "4.5", "4"])
s3 = s2.astype(float)
print(s2.dtype, s3.dtype,
      end='\n----\n')
print(s1.tolist(),
      s1.unique().tolist(),
      sep='\n',
      end='\n----\n')
print(s1.value_counts(),
      end='\n----\n')
print(s1.factorize(),
      end='\n----\n')
print(s1.map(pariedispari),
      end='\n----\n')
```

# Esempio

---

```
0 => 3
8 => 6
8 => 6
2 => 3
0 => 3
3 => 3
1 => 3
0 => 3
object float64
-----
[0, 8, 8, 2, 0, 3, 1, 0]
[0, 8, 2, 3, 1]
-----
0      3
8      2
2      1
3      1
1      1
dtype: int64
-----
(array([0, 1, 1, 2, 0, 3, 4, 0], dtype=int64), Int64Index([0, 8, 2, 3, 1], dtype='int64'))
-----
0      pari
1      pari
2      pari
3      pari
4      pari
5      dispari
6      dispari
7      pari
dtype: object
-----
```

# DataFrame



# DataFrame

---

- Un DataFrame pandas è una struttura dati tabellare bidimensionale mutabile in dimensioni, potenzialmente eterogenea con assi etichettati (righe e colonne).
- Un DataFrame è costituito da tre componenti principali
  - i dati
  - le righe
  - le colonne.
- Un Dataframe è l'unione di più Series

# DataFrame

---

- Normalmente, un DataFrame Pandas viene creato caricando i set di dati da una archiviazione esistente,
  - un database SQL,
  - un file CSV
  - un file Excel.
  - ...
- Un DataFrame può essere creato anche da
  - elenchi, *liste*
  - dizionari
  - da un elenco di dizionari,
  - ecc. *listo*

# dataclasses

---

- Alcune volte può tornare utile la libreria `dataclasses`
- La libreria permette di creare delle `data class`
- Struttura dati di tipo classe con dichiarazioni di tipo

```
from dataclasses import make_dataclass
import pandas as pd

Point = make_dataclass("Point", [("x", int), ("y", int)])
df1 = pd.DataFrame([Point(0, 0), Point(0, 3), Point(2, 3)])
print(df1)
```

# DataFrame

---

- Uso di liste:

```
a_list = [[ "Massimo", "Giovanni", "Francesco" ],  
          [ "Regoli", "Trapattoni", "Totti" ],  
          [ "Ricercatore", "Allenatore", "Attaccante" ]]  
df2 = pd.DataFrame(a_list)  
df2.columns = [ "Nome", "Cognome", "Attività" ]
```

# DataFrame

- Attenzione!

```
In [95]: 1 v =[['A', 1, True], ['B', 2, False], ['C', 3], [None, '3']]  
2 df = pd.DataFrame(v)  
3 df
```

Out[95]:

	0	1	2
0	A	1	True
1	B	2	False
2	C	3	None
3	None	3	None

Dati mancanti  
sostituiti da  
None

# DataFrame

---

- Uso di dict

```
a_dict = {"Nome": ["Massimo", "Giovanni", "Francesco"],  
          "Cognome": ["Regoli", "Trapattoni", "Totti"],  
          "Attività": ["Ricercatore", "Allenatore", "Attaccante"]}  
df1 = pd.DataFrame(a_dict)  
df1
```

	Nome	Cognome	Attività
0	Massimo	Regoli	Ricercatore
1	Giovanni	Trapattoni	Allenatore
2	Francesco	Totti	Attaccante

Le chiavi del  
dizionario possono  
essere usate per  
nominare le colonne  
del Dataframe

# Dataframe

---

- Uso di `dict` e `from_dict`

```
df2 = pd.DataFrame.from_dict(  
    dict([("A", [1, 2, 3]), ("B", [4, 5, 6])]))  
print(df2)
```

```
df3 = pd.DataFrame.from_dict(  
    dict([("A", [1, 2, 3]), ("B", [4, 5, 6])])), orient="index")  
print(df3)
```

	A	B
0	1	4
1	2	5
2	3	6

	0	1	2
A	1	2	3
B	4	5	6

# DataFrame

- Attenzione!

```
In [99]: 1 d = {"Col1" : [1, 2, 3, 4], "Col2" : ['a', 'b', 'c', 'd'],
           2      "Col3": [True, False, False]}
           3 df = pd.DataFrame(d)
           4 df
```

---

```
-----  
ValueError                                     Traceback (most recent call last)  
<ipython-input-99-2a2306003340> in <module>  
      1 d = {"Col1" : [1, 2, 3, 4], "Col2" : ['a', 'b', 'c', 'd'],  
      2      "Col3": [True, False, False]}  
----> 3 df = pd.DataFrame(d)  
      4 df
```

La  
dimensione  
delle liste  
deve essere,  
in questo  
caso,  
omogenea

# DataFrame – Load

---

- Lettura file:
  - `read_csv`
    - Crea un DataFrame da file csv
  - `read_json`
    - Crea un DataFrame da file json
  - `read_sql`
    - Crea un DataFrame da tabella sql (utile sqlalchemy)

1, 2, NA

2, 3, 3

4, 5, 2

3, 2, 'e', 7.5

# DataFrame

---

- Uso di csv

- sep: str default ","
- Vedere: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

```
import requests
conn = requests.get("https://raw.githubusercontent.com" +
                     "/italia/covid19-opendata-vaccini/master" +
                     "/dati/somministrazioni-vaccini-latest.csv")
csv_data = conn.text
f = open("vaccini.csv", "w")
f.write(csv_data)
f.close()
vacc_data = pd.read_csv("vaccini.csv", encoding='latin-1')
vacc_data
```

# DataFrame

- Ma anche:

```
df = pd.read_csv("https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-regioni/dpc-covid19-ita-regioni.csv")
df
```

	data	stato	codice_regione	denominazione_regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isola
0	2020-02-24T18:00:00	ITA	13	Abruzzo	42.351222	13.398438	0	0	0	0
1	2020-02-24T18:00:00	ITA	17	Basilicata	40.639471	15.805148	0	0	0	0
2	2020-02-24T18:00:00	ITA	4	P.A. Bolzano	46.499335	11.356624	0	0	0	0
3	2020-02-	ITA	18	Calabria	38.905076	16.594402	0	0	0	0

# DataFrame

- Uso di json

- See: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_json.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_json.html)



```
1 conn = requests.get("https://raw.githubusercontent.com/\"\
2                         \"pcm-dpc/COVID-19/master/dati-json/\"\
3                         \"dpc-covid19-ita-regioni.json")  
4 jsondata = conn.text  
5  
6 df2 = pd.read_json(jsondata)  
7 df2
```

# DataFrame

- Ma anche:

```
df = pd.read_json("https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-json/dpc-covid19-ita-regioni.json")
df
```

	data	stato	codice_regione	denominazione_regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isola
0	2020-02-24T18:00:00	ITA	13	Abruzzo	42.351222	13.398438	0	0	0	0
1	2020-02-24T18:00:00	ITA	17	Basilicata	40.639471	15.805148	0	0	0	0
2	2020-02-24T18:00:00	ITA	4	P.A. Bolzano	46.499335	11.356624	0	0	0	0
3	2020-02-24T18:00:00	ITA	18	Calabria	38.905976	16.594402	0	0	0	0

# DataFrame

- Uso sql

```
dflazio = pd.read_sql("SELECT * FROM covid19 WHERE denominazione_regione='Lazio'", engine)  
dflazio
```

engine  
potrebbe  
essere una  
connessione ad un dbl  
sqlite

	data	stato	codice_regione	denominazione_regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale
0	2020-02-24T18:00:00	ITA	12	Lazio	41.89277	12.483667	1	1	1
1	2020-02-25T18:00:00	ITA	12	Lazio	41.89277	12.483667	1	1	1
2	2020-02-26T18:00:00	ITA	12	Lazio	41.89277	12.483667	0	0	0
3	2020-02-27T18:00:00	ITA	12	Lazio	41.89277	12.483667	0	0	0

# DataFrame

---

- Esistono anche i metodi duali:
  - `to_csv`
    - Scrive un DataFrame su un file csv
  - `to_json`
    - Scrive un DataFrame su un file json
  - `to_sql`
    - Scrive un database (necessario sqlalchemy)

# DataFrame

---

- È possibile leggere inoltre:
  - Excel
  - HTML
  - Stata
  - SPSS
  - SAS

# DataFrame

---

- Un DataFrame è una struttura di dati bidimensionale, ovvero i dati sono allineati in modo tabellare in righe e colonne.
- Possiamo eseguire operazioni di base su righe/colonne come
  - selezionare,
  - eliminare,
  - aggiungere
  - rinominare.

# DataFrame

- Selezione colonna:

- per selezionare una colonna possiamo accedere alle colonne chiamandole con il nome (o l'indice).
- Notare la doppia [...] necessaria per accedere a più colonne

```
| 1 df2[["data", "denominazione_regione", "ricoverati_con_sintomi"]]
```

```
:
```

	data	denominazione_regione	ricoverati_con_sintomi
0	2020-02-24T18:00:00	Abruzzo	0
1	2020-02-24T18:00:00	Basilicata	0
2	2020-02-24T18:00:00	P.A. Bolzano	0
3	2020-02-24T18:00:00	Calabria	0
4	2020-02-24T18:00:00	Campania	0
...	...	...	...
1297	2020-04-25T17:00:00	Toscana	687
1298	2020-04-25T17:00:00	P.A. Trento	202
1299	2020-04-25T17:00:00	Umbria	95
1300	2020-04-25T17:00:00	Valle d'Aosta	83
1301	2020-04-25T17:00:00	Veneto	1105

1302 rows × 3 columns

# DataFrame

---

- Selezione Riga
- Come per Series è possibile accedere a righe assolute, per indice o per nome
  - [ ] per le colonne
  - .loc[ ] posizione indicizzata
  - .iloc[ ] posizione assoluta
- Inoltre è possibile usare gli operatori di filtraggio booleano

# Esempio

```
1 lazio = df2["denominazione_regione"] == "Lazio"
2 df2[lazio][["data", "ricoverati_con_sintomi"]]
```

	data	ricoverati_con_sintomi
7	2020-02-24T18:00:00	1
28	2020-02-25T18:00:00	1
49	2020-02-26T18:00:00	0
70	2020-02-27T18:00:00	0
91	2020-02-28T18:00:00	0
...	...	...
1204	2020-04-21T17:00:00	1380
1225	2020-04-22T17:00:00	1384
1246	2020-04-23T17:00:00	1385
1267	2020-04-24T17:00:00	1396
1288	2020-04-25T17:00:00	1421

62 rows × 2 columns

# DataFrame

---

- Conversione in liste

```
1 | y = dati_lazio["ricoverati_con_sintomi"].to_list()  
2 | x = dati_lazio["data"].to_list()
```

```
1 | print(x[0], y[0])
```

2020-02-24T18:00:00 1

# DataFrame – metodi

FUNCTION	DESCRIPTION
<code>index</code>	Method returns index (row labels) of the DataFrame
<code>insert()</code>	Method inserts a column into a DataFrame
<code>nunique()</code>	Method returns count of the unique values in the dataframe
<code>value_counts()</code>	Method counts the number of times each unique value occurs within the Series
<code>columns</code>	Method returns the column labels of the DataFrame
<code>axes</code>	Method returns a list representing the axes of the DataFrame
<code>isin()</code>	Method extracts rows from a DataFrame where a column value exists in a predefined collection
<code>sort_values()</code>	Method sorts a data frame in Ascending or Descending order of passed Column
<code>loc[]</code>	Method retrieves rows based on index label
<code>iloc[]</code>	Method retrieves rows based on index position

# Esempio

---

```
df = pd.DataFrame({  
    'specie': ['falcon', 'dog', 'cat', 'ant', 'spider', 'butterfly', 'human'],  
    'num_legs': [2, 4, 4, 6, 8, 6, 2],  
    'num_wings': [2, 0, 0, 0, 0, 4, 0],  
})  
print(df.nunique())  
print(df.axes)  
bipeds = df['num_legs'].isin([2])  
print(df[bipeds])
```

# Esempio

```
col1  col2  col3  col4
0     A      2      0    a
1     A      1      1    B
2     B      9      9    c
3    NaN      8      4    D
4    pd.DataFrame({7'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
5      'col2': [2, 1, 0, 8, 7, 4],
6      'col3': [0, 1, 9, 4, 2, 3],
7      'col4': [1, 8, 3, 5, 6, 9]},
8      index=[0, 1, 2, 3, 4, 5])
print(df.sort_values(by=['col1']))
print(df.sort_values(by=['col1', 'col3']))
print(df.sort_values(by=['col4'], ascending=False))
```

	col1	col2	col3	col4
0	A	2	0	a
1	A	1	1	B
2	B	9	9	c
3	NaN	8	4	D
4				
5				
6				
7				
8				
9				

	col1	col2	col3	col4
0	A	1	1	B
1	A	2	0	a
2	B	9	9	c
3	NaN	8	4	D
4	C	4	3	F
5				
6				
7				
8				
9				

Vari tipi di ordinamento, ma attenzione all'ultimo!  
La quarta colonna potrebbe non essere stata ordinata come la desideriamo

# Esempio

---

```
0    A    2    0    a  
1    A    1    1    B  
2    B    9    9    c  
3   NaN    8    4    D  
4    D    7    2    e  
5    C    4    3    F
```

La funzione lambda permette di indicare la trasformazione da fare prima dell'ordinamento.  
N.B. i dati non vengono cambiati nel DataFrame

```
print(df.sort_values(by='col4', key=lambda col: col.str.lower()))
```

# DataFrame – metodi

---

FUNCTION	DESCRIPTION
<code>rename()</code>	Method is called on a DataFrame to change the names of the index labels or column names
<code>columns</code>	Method is an alternative attribute to change the column name
<code>drop()</code>	Method is used to delete rows or columns from a DataFrame
<code>nsmallest()</code>	Method pulls out the rows with the smallest values in a column
<code>nlargest()</code>	Method pulls out the rows with the largest values in a column
<code>query()</code>	Method is an alternate string-based syntax for extracting a subset from a DataFrame
<code>copy()</code>	Method creates an independent copy of a pandas object

# Esempio

---

```
import pandas as pd

wh = pd.read_csv("weight-height.csv")
print(wh.columns)
wh.rename(columns={"Height": "Inches",
                   "Weight": "Pound"},  
         inplace=True)
print(wh.columns)

print(df.nsmallest(2, 'col2'))
print(df.nlargest(2, 'col3'))
```

inplace applica la trasformazione all'oggetto corrente

	col1	col2	col3	col4
1	A	1	1	B
0	A	2	0	a
	col1	col2	col3	col4
2	B	9	9	c
3	NaN	8	4	D

# DataFrame – metodi

---

```
col2  col3
0     2     0
1     1     1
2     9     9
3     8     4
4     7     2
5     4     3

print(wh.query('Gender == "Male" and Inches < 60'))
print(df.drop(columns=['col1', 'col4']))
```

	Gender	Inches	Pound
1942	Male	59.981865	112.902939
2191	Male	59.938650	141.459579
2334	Male	59.380650	136.391006
4637	Male	59.868078	117.803842
4794	Male	58.406905	121.338323

# DataFrame

- Esempi per query
- Il metodo query permette di creare un filtro simile ad una query SQL

```
newdf[newdf["ricoverati_con_sintomi"] < newdf["terapia_intensiva"]]
```

		data	regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isolamento_domiciliare	totale_positivi	variaz
400		2020-03-14T17:00:00	Basilicata	40.639471	15.805148	0	2	2	8	10	
421		2020-03-15T17:00:00	Basilicata	40.639471	15.805148	0	2	2	9	11	
431		2020-03-15T17:00:00	Molise	41.557748	14.659161	3	4	7	10	17	
442		2020-03-16T17:00:00	Basilicata	40.639471	15.805148	1	2	3	9	12	
452		2020-03-16T17:00:00	Molise	41.557748	14.659161	3	5	8	7	15	

```
newdf.query('ricoverati_con_sintomi < terapia_intensiva & totale_positivi > 12')
```

		data	regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isolamento_domiciliare	totale_positivi	variaz
431		2020-03-15T17:00:00	Molise	41.557748	14.659161	3	4	7	10	17	
452		2020-03-16T17:00:00	Molise	41.557748	14.659161	3	5	8	7	15	

# DataFrame – Trasformazioni

---

- Un DataFrame è un oggetto manipolabile non solo attraverso gli slicing orizzontali o verticali
  - Filtri, Proiezioni
- Ma anche per la possibilità di aggiungere colonne
  - Standalone
  - Calcolate

# Trasformazioni

---

```
wh[ "Kilos" ] = wh[ "Pound" ] * 0.453592  
wh[ "Meters" ] = wh[ "Inches" ] * 0.0254  
wh[ "BMI" ] = wh[ "Kilos" ] / (wh[ "Meters" ] ** 2)  
print(wh.iloc[0:5])
```

	Gender	Inches	Pound	Kilos	Meters	BMI
0	Male	73.847017	241.893563	109.720985	1.875714	31.185761
1	Male	68.781904	162.310473	73.622732	1.747060	24.121044
2	Male	74.110105	212.740856	96.497550	1.882397	27.232906
3	Male	71.730978	220.042470	99.809504	1.821967	30.067059
4	Male	69.881796	206.349801	93.598619	1.774998	29.708033

# Trasformazioni

---

	Gender	Inches	Pound	Kilos	Meters	BMI
86	M	66.596197	208.345694	94.503940	1.691543	33.028103
3524	M	66.145868	202.130274	91.684675	1.680105	32.480588
727	M	63.400423	185.188138	83.999858	1.610371	32.391183
1671	M	70.059331	225.014368	102.064717	1.779507	32.231182
3061	M	69.219568	218.314451	99.025689	1.758177	32.034848

```
wh["Gender"] = wh["Gender"].transform(lambda g: g[0])
print(wh.nlargest(5, "BMI"))
```

# DataFrame

---

- `pandas.cut`:
  - Trasforma valori in intervalli discreti (categorie).
  - Utilizzare `cut` quando è necessario segmentare e ordinare i valori dei dati in categorie.
  - Questa funzione è utile per passare da una variabile continua a una variabile categoriale.
  - Ad esempio, il taglio (`cut`) potrebbe convertire le età in gruppi di fasce di età.

# Pandas cut

---

	Gender	Inches	Pound	Kilos	Meters	BMI	Cat
8795	F	66.419316	151.717651	68.817913	1.687051	24.179386	OK
7584	F	64.740126	147.269670	66.800344	1.644399	24.703824	OK
8078	F	57.714257	94.389733	42.814428	1.465942	19.923081	-
5967	F	63.350342	136.181917	61.771028	1.609099	23.857200	OK
3643	M	70.207802	194.205523	88.090071	1.783278	27.700575	+

```
wh["Cat"] = pd.cut(wh["BMI"], 5, labels=["--", "-", "OK", "+", "++"])
print(wh.sample(5))
```

# Pandas groupby

---

- La funzione `dataframe.groupby()` viene utilizzata per suddividere i dati in gruppi in base ad alcuni criteri.
- Gli oggetti pandas possono essere divisi su uno qualsiasi dei loro assi.
- La definizione astratta di raggruppamento consiste nel fornire una mappatura delle etichette ai nomi dei gruppi.

# Esempio

---

```
print(wh.groupby(["Gender"])[["Kilos", "Meters"]].mean())
```

Gender	Kilos	Meters
F	61.625051	1.618203
M	84.831057	1.753269

# DataFrame

---

- **pivot\_table:**

- Crea una tabella pivot in stile foglio di calcolo come DataFrame.
- I livelli nella tabella pivot verranno organizzati in indici (MultiIndex gerarchici) e colonne DataFrame (risultato).

# pandas pivot\_table

```
1 df = pd.read_json("https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-json/dpc-covid19-ita-regioni.json")
2 pd.pivot_table(df, index="data", values=["nuovi_positivi","totale_positivi"], aggfunc=sum).tail(n=10)
```

data	nuovi_positivi	totale_positivi
2020-04-28T17:00:00	2091	105205
2020-04-29T17:00:00	2086	104657
2020-04-30T17:00:00	1872	101551
2020-05-01T17:00:00	1965	100943
2020-05-02T17:00:00	1900	100704
2020-05-03T17:00:00	1389	100179
2020-05-04T17:00:00	1221	99980
2020-05-05T17:00:00	1075	98467
2020-05-06T17:00:00	1444	91528
2020-05-07T17:00:00	1401	89624

# Funzioni Statistiche di base

---

- **pct\_change**
  - Questa funzione confronta ogni elemento con il suo elemento precedente e calcola la percentuale di variazione.
- **cov**
  - Calcola la covarianza tra gli oggetti della serie.
  - NA verrà escluso automaticamente.
- **corr**
  - Esistono diversi metodi per calcolare la correlazione
    - pearson (impostazione predefinita),
    - spearman
    - kendall.

```
1 df["tamponi"].corr(df["totale_positivi"])
0.7848375109955518
```

# Correlazione

```
1 dfLo.corr()
```

	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isolamento_domiciliare	totale_positivi
verati_con_sintomi	1.000000	0.946493	0.999544	0.561479	0.774250
terapia_intensiva	0.946493	1.000000	0.955803	0.304105	0.559871
totale_ospedalizzati	0.999544	0.955803	1.000000	0.539960	0.757720
isolamento_domiciliare	0.561479	0.304105	0.539960	1.000000	0.958409
totale_positivi	0.774250	0.559871	0.757720	0.958409	1.000000
one_totale_positivi	0.116287	0.261202	0.130370	-0.264155	-0.160599
nuovi_positivi	0.568226	0.725949	0.585563	-0.027265	0.177413
dimessi_guariti	0.520211	0.256897	0.497949	0.982129	0.930308
deceduti	0.623770	0.364472	0.602355	0.987024	0.969505
totale_casi	0.667122	0.423515	0.647372	0.986869	0.984649
totale_tamponi	0.407076	0.140583	0.384002	0.974946	0.886102

# DataFrame

- Sorting data
  - Un DataFrame può essere ordinato per una o più colonne o per indice

```
1 (dfLo[["data", "deceduti", "totale_positivi"]]
2     .sort_values(
3         by=["deceduti", "totale_positivi"],
4         ascending=[True, False])).head(5)
```

		data	deceduti	totale_positivi
<b>9</b>		2020-02-24T18:00:00	6	166
<b>51</b>		2020-02-26T18:00:00	9	249
<b>30</b>		2020-02-25T18:00:00	9	231
<b>72</b>		2020-02-27T18:00:00	14	349
<b>93</b>		2020-02-28T18:00:00	17	474

