Lezione 3: Plotting

Davide Evangelista
davide.evangelista5@unibo.it

Università di Bologna

26 Febbraio 2024

Plot Introduzione

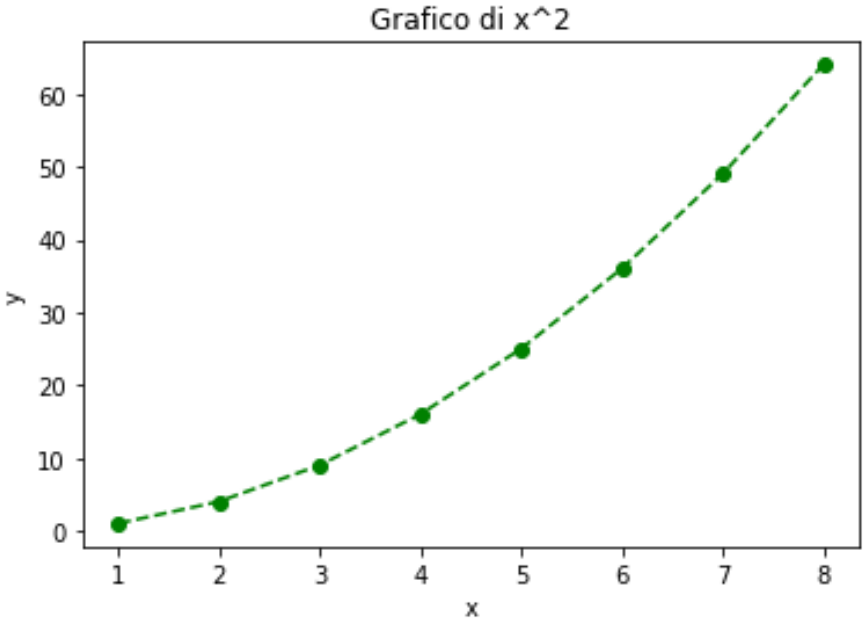
- Per rappresentare i vettori delle osservazioni e controllare come sono distribuiti i dati nel nostro dataset, usiamo Matplotlib.
- Si importa con il comando `import matplotlib.pyplot as plt`

La funzione `plot()` prende come input molti argomenti:

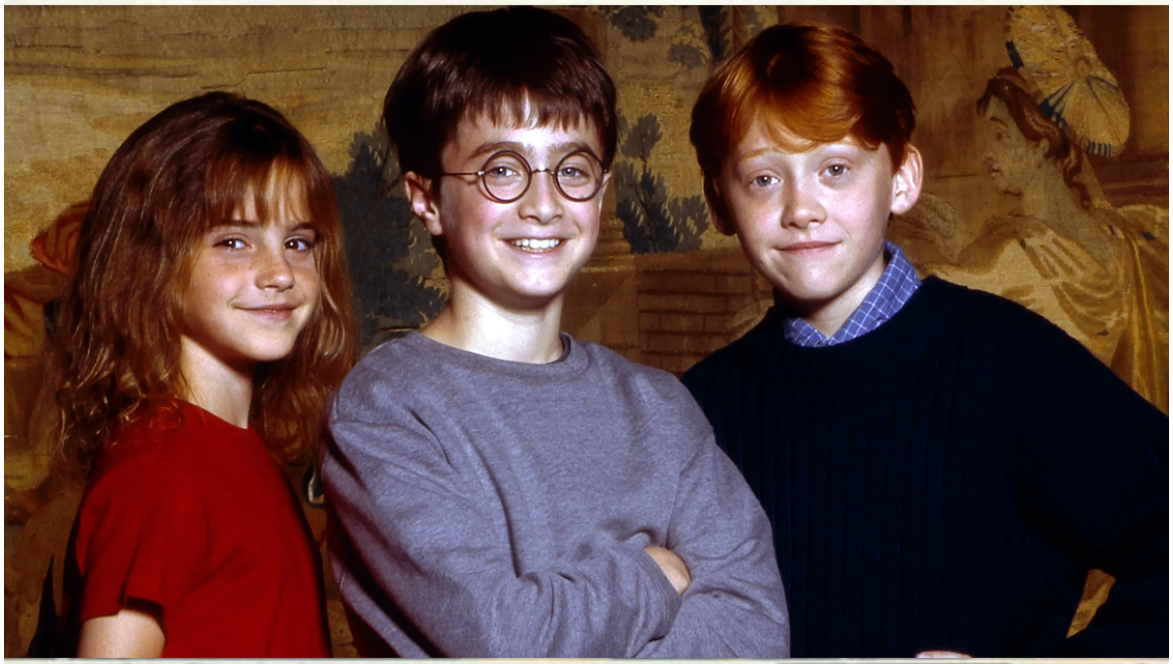
- Un vettore `x` rappresentante le ascisse dei punti da plottare.
- Un vettore `y` rappresentante le ordinate dei punti da plottare.
- `color` che indica il colore del grafico.
- `marker` che indica il marker che rappresenta i punti. [ref]
- `linestyle` che indica il tipo di linea che vogliamo usare.[ref]

Plot Esempio

```
> x = np.arange(1,9)
> y = x**2
> plot(x, y, color='green', marker='o', linestyle='dashed')
> plt.xlabel('x')
> plt.ylabel('y')
> plt.title('Grafico di x^2')
> plt.show()
```



Name A Better Trio. I'll Wait 😴



```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
```

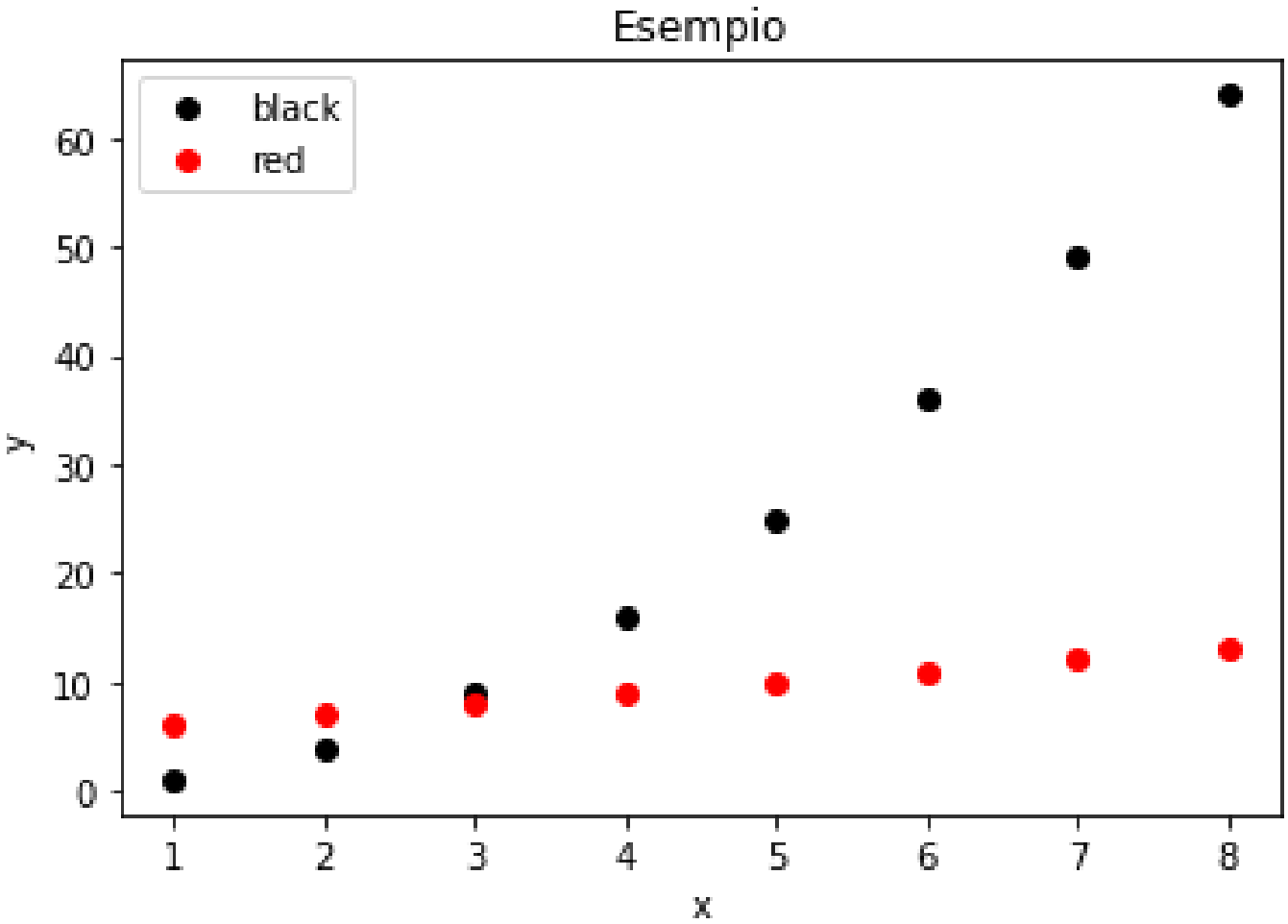
That's it.

Plot(s)

- La funzione `plot()` disegna il nuovo grafico nella stessa figura.

```
> x = np.arange(1,9)
> y = x**2
> y1 = x + 5
> plt.plot(x, y, 'ko')
> plt.plot(x,y1,'ro')
> plt.xlabel('x')
> plt.ylabel('y')
> plt.legend(['black','red'])
> plt.title('Esempio')
> plt.show()
```

Plot(s)



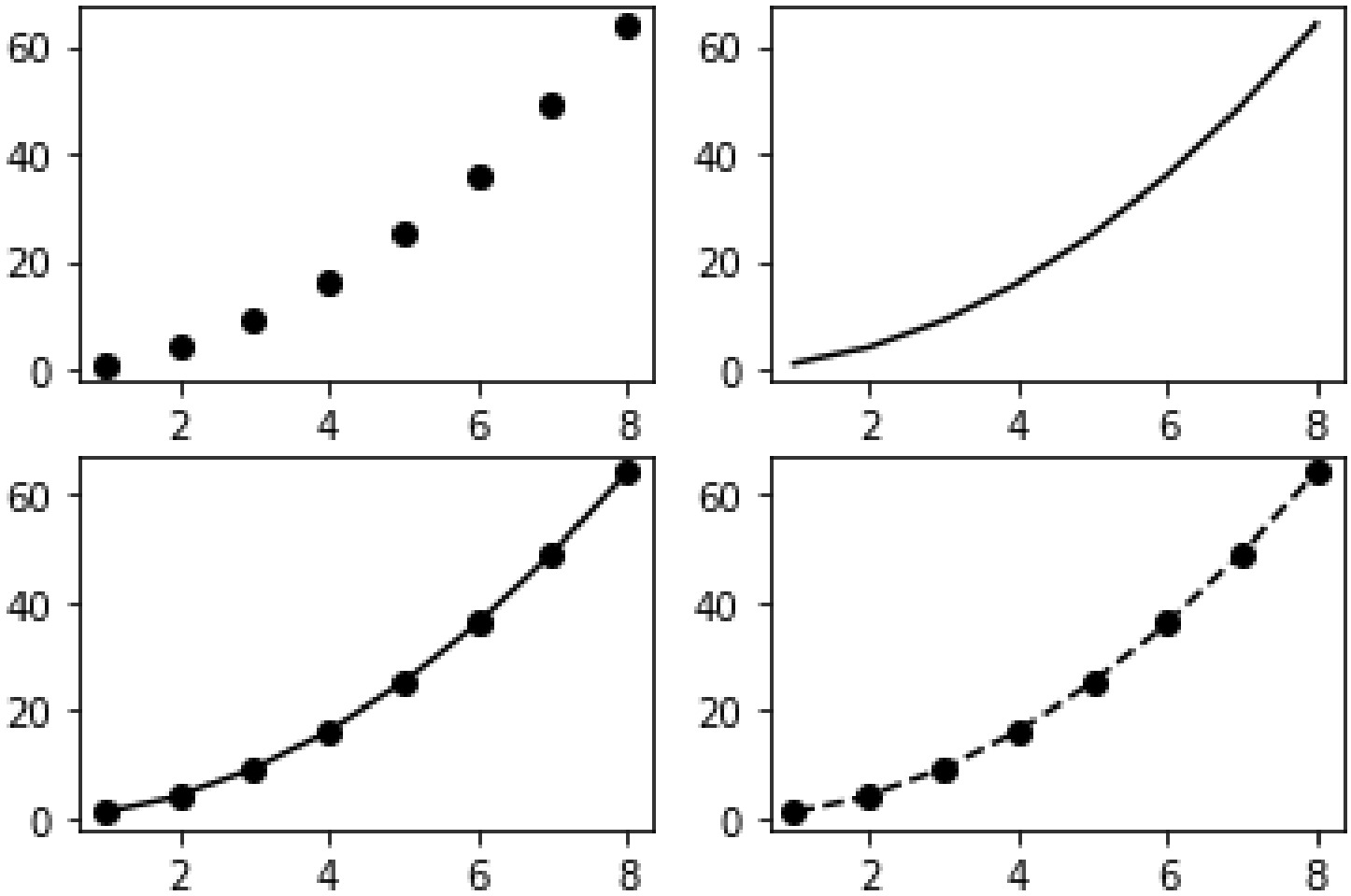
Subfigure

- Per disegnare più plot nella stessa schermata si usa la funzione `plt.subplots(nrows=rnum, ncols=cnum)`, dove `rnum` rappresenta il numero di plot che si vuole visualizzare in riga, mentre `cnum` rappresenta il numero di plot che si vuole visualizzare in colonna.

```
> fig, ax = plt.subplots(nrows=2, ncols=2)
> ax[0,0].plot(x, y, 'ko')
> ax[0,1].plot(x, y, 'k-')
> ax[1,0].plot(x, y, 'k-o')
> ax[1,1].plot(x, y, 'k--o')

> plt.show()
```

Subfigure

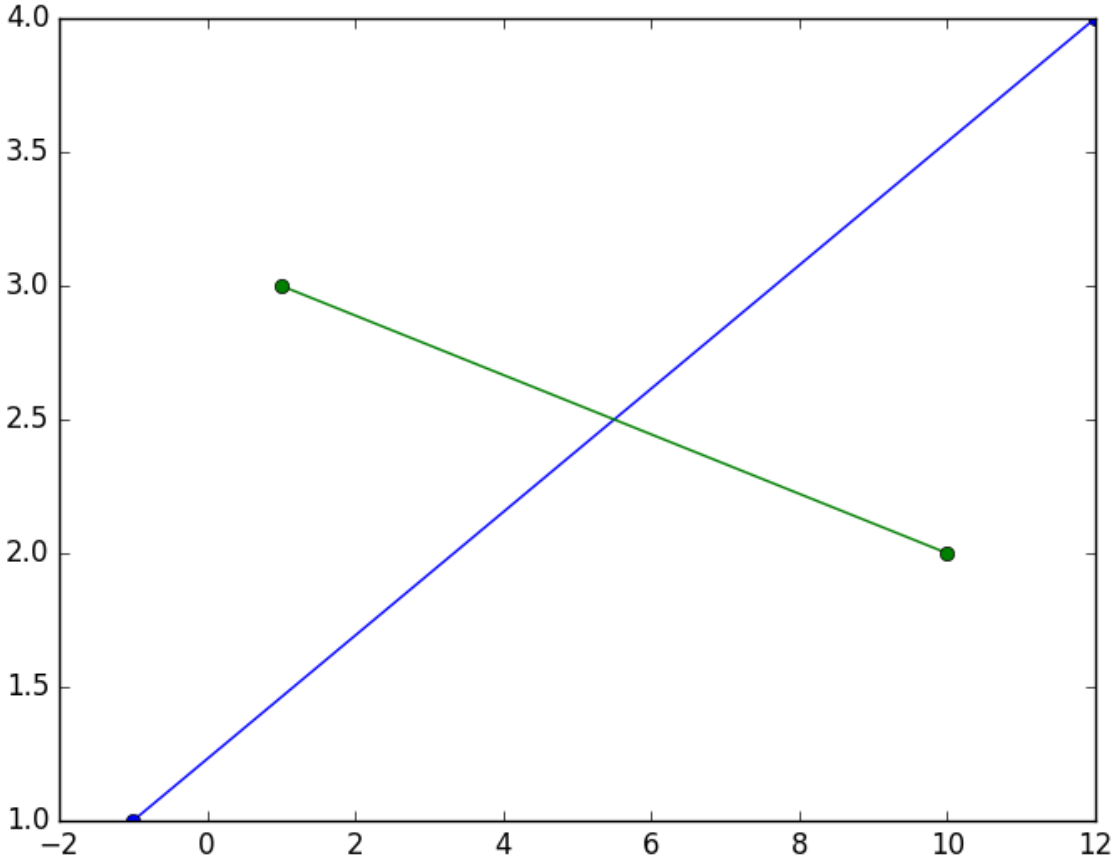


Segmenti e linee

- Per disegnare una retta si usa il comando `axline(xy1, xy2=None, slope=None)`
  - `xy1, xy2` punti per cui passa la retta.
  - `slope` coefficiente angolare retta (se non si specifica `xy2`)
- Si possono disegnare i segmenti prendendo in input le coordinate dei due punti estremi.

```
> x1, y1 = [-1, 12], [1, 4]
> x2, y2 = [1, 10], [3, 2]
> plt.plot(x1, y1, x2, y2, marker = 'o')
> plt.show()
```

Segmenti e linee



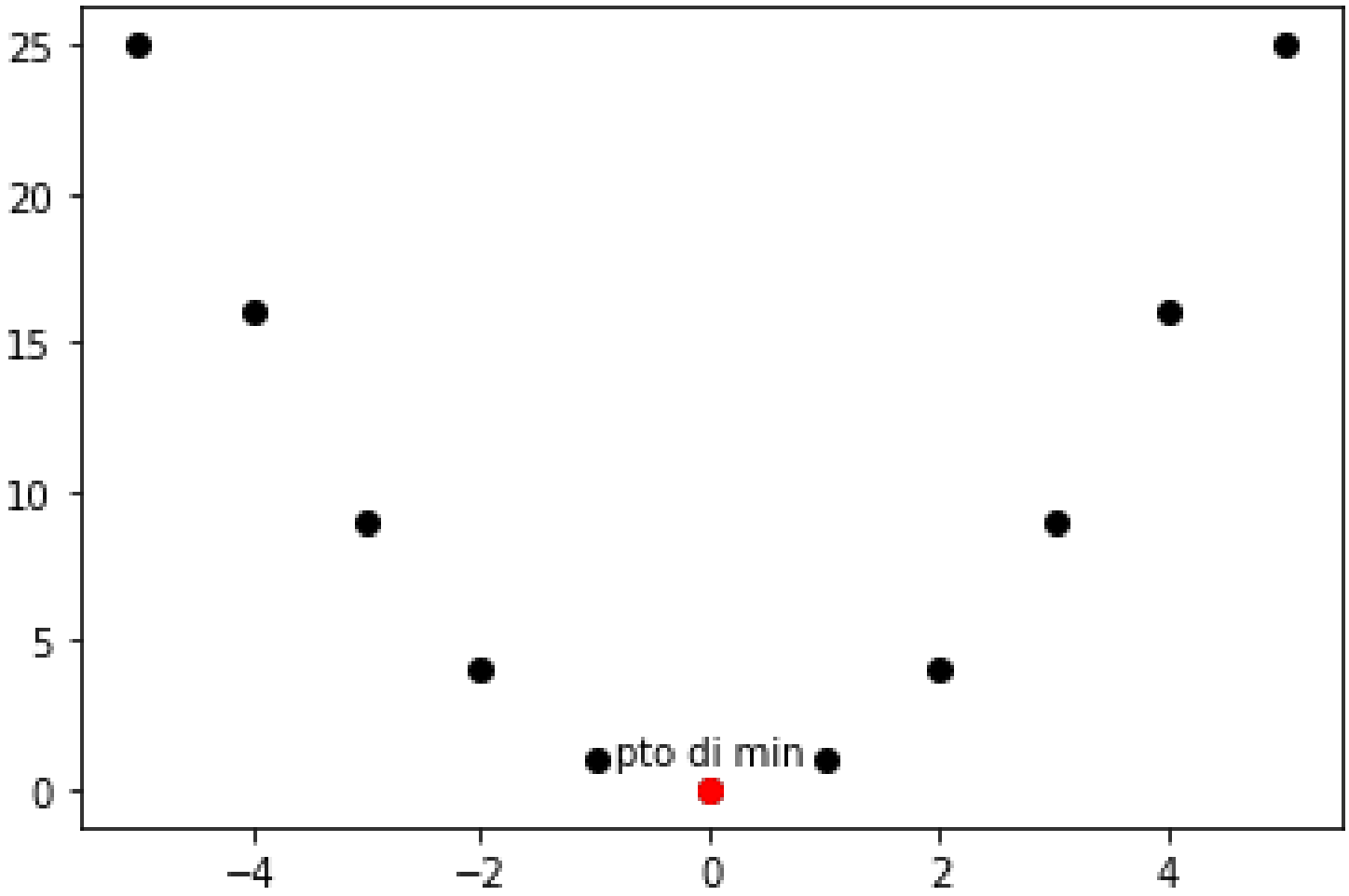
Text

La funzione `text()` serve per aggiungere dei label sul grafico.
Utile quando si vuole aggiungere il nome ad uno specifico punto del grafico.

```
> x = np.arange(-5,6)
> y = x**2

> plt.plot(x,y,'ko')
> plt.plot(0,0,'ro')
> plt.text(0,0.8, 'pto di min', horizontalalignment='center')
> plt.show()
```

Text



Estrazioni casuali: `random.choice()`

- Dato un vettore `v` la funzione `random.choice()` estrae da `v` (con reinserimento) *n* valori.

```
> v = np.arange(1,11)
> np.random.choice(v,10)
[1] 2 9 5 9 1 9 2 6 5 9
```

- Per estrarli con reinserimento

```
> np.random.choice(v,8, replace=False)
[1] 2 9 1 7 3 6 8 4
```

- È possibile specificare la probabilità di estrarre ogni singolo elemento.

```
> v = np.arange(1,4)
> np.random.choice(v, 10, replace=True, p=(0.7 , 0.2 , 0.1
[1] 1 1 2 3 1 1 2 3 1 1
```

Estrazioni casuali: normale Gaussiana

La funzione `random.normal()` prende in input

- la media
- la deviazione standard
- la lunghezza del vettore di output

restituisce un vettore che ha dimensione scelta e contiene elementi estratti con una distribuzione Gaussiana con media e deviazione assegnate.

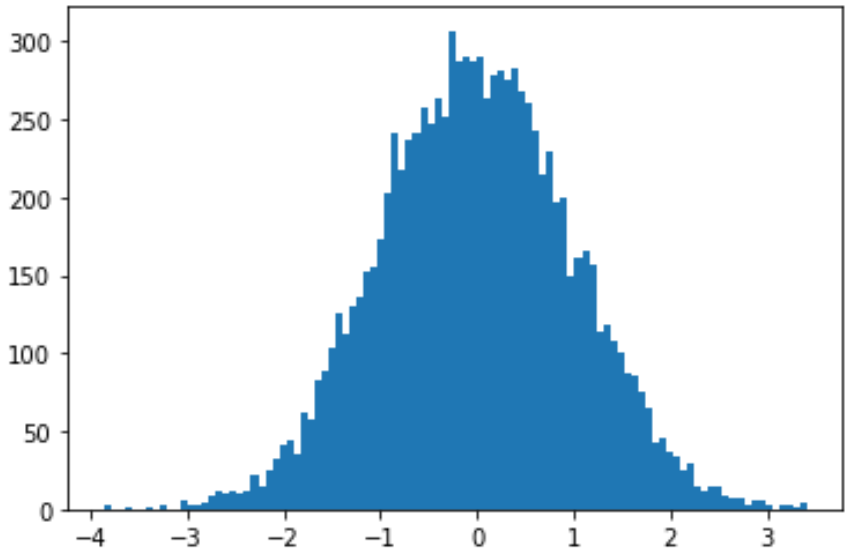
```
> x = random.normal(loc=0.0, scale=1.0, size=10)
> x # Normale standard
[1] 0.35315767 0.81645816 0.27591868 -1.09573384
1.63935188 0.80787089 -0.07381423 0.63730397
-0.61066481 -0.89062814
```

- texttttrandom.randn() stessa cosa senza specificare media e deviazione standard (0,1).

Istogrammi

Per disegnare un istogramma si utilizza la funzione `hist()` che prende come input un vettore (del quale plottare le frequenze) e un parametro opzionale `bins`, che indica in quanti intervalli dividere i valori dell'array in input.

```
> x = np.random.randn(10000)
> plt.hist(x , bins=100)
```



Barplot & Piechart

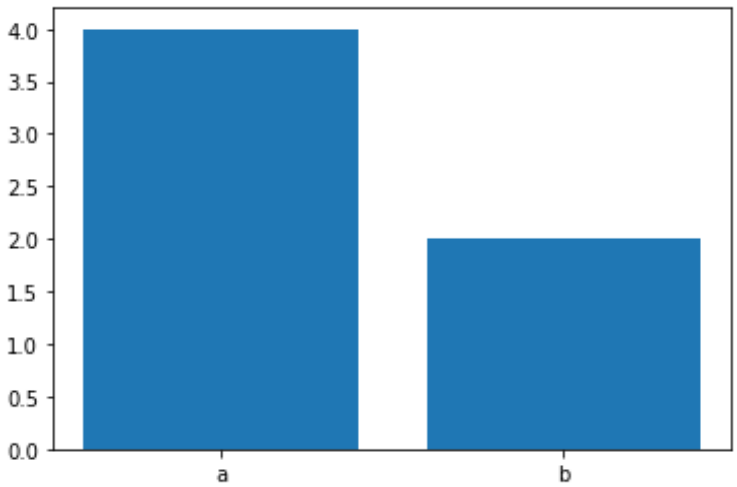
Per rappresentare un vettore di tipo qualitativo si utilizzano le seguenti funzioni

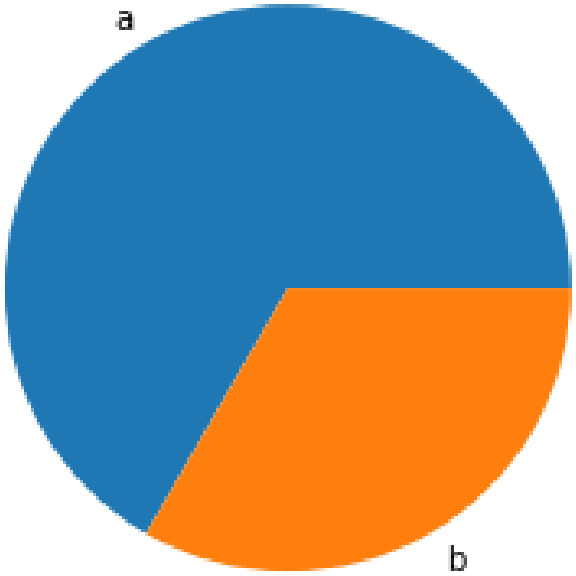
- `bar()` per rappresentare dei grafici a barre
- `piechart()` per rappresentare dei grafici a torta

Prima è necessario contare le occorrenze di ogni possibile valore.

```
> x = np.array(("a" , "a" , "a" , "b" , "a" , "b" ))
> unique = np.unique(x)
> count = [np.sum(x==el) for el in unique]
> plt.bar(unique,count)
> plt.pie(count,labels=unique)
```

Barplot & Piechart

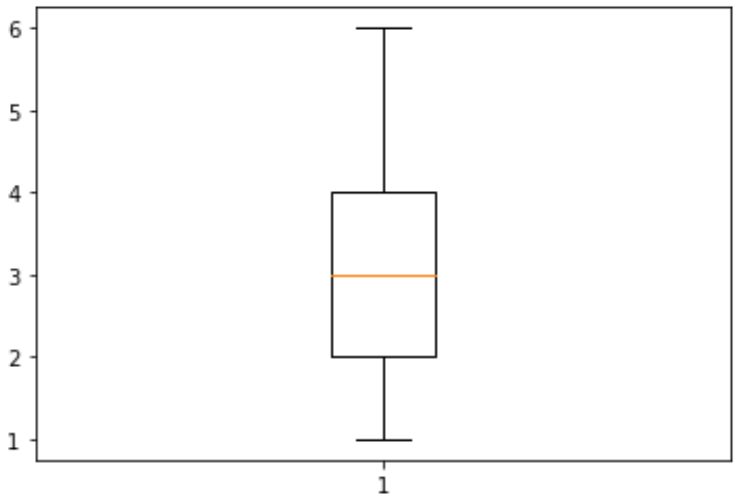




Boxplot

Il `boxplot()`, risulta utile per visualizzare il range dei dati.

```
> x = np.array((3, 3, 3, 2, 1, 4, 6, 2, 4, 1, 6, 4, 2, 1, 5, 4
> plt.boxplot(x)
```



Seaborn

- Plot con Pyplot molto flessibili ma la sintassi è molto verbosa.
- La libreria Seaborn permette di creare grafici più complessi.
- Ottima integrazione con i DataFrame in Pandas
- Per una lista completa dei possibili grafici si veda la Python-Graph Gallery.

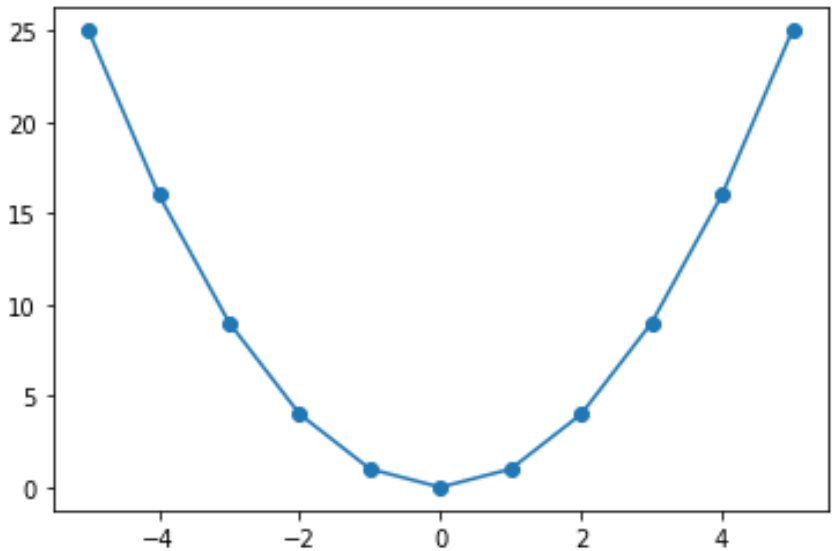
```
import seaborn as sns
```


MatplotLib
Seaborn
imgflip.com
JAKE-CLARK.TUMBLR

Esempio Seaborn

Disegnamo il grafico della funzione $f(x) = x^2$ sull'intervallo $[-5, 5]$.
Per passare i dati a ggplot2 è necessario salvarli in un dataframe.

```
> df = pd.DataFrame({'x_axis': np.arange(-5,6),
'y_axis': np.arange(-5,6)**2})
> plt.plot('x_axis', 'y_axis',
data=df, linestyle='-', marker='o')
> plt.show()
```



Esempio Seaborn

```
# load data
> df = sns.load_dataset('iris')
# plot
> sns.violinplot(x=df["species"], y=df["sepal_length"])
```