lez_09

February 2, 2026

1 Matplotlib

Matplotlib è una libreria Python utilizzata per creare visualizzazioni di dati statiche, animate e interattive. È basata su NumPy ed è in grado di gestire facilmente grandi set di dati per creare vari tipi di grafici, come grafici a linee, grafici a barre, grafici a dispersione, ecc.

1.1 Pyplot

Pyplot è un modulo di Matplotlib che fornisce un'interfaccia semplice per la creazione di grafici. Consente agli utenti di generare grafici come grafici a linee, grafici a barre e istogrammi con un codice minimo. Esploriamo alcuni esempi con codice semplice per capire come utilizzarlo in modo efficace.

1.1.1 1. Grafico a linea

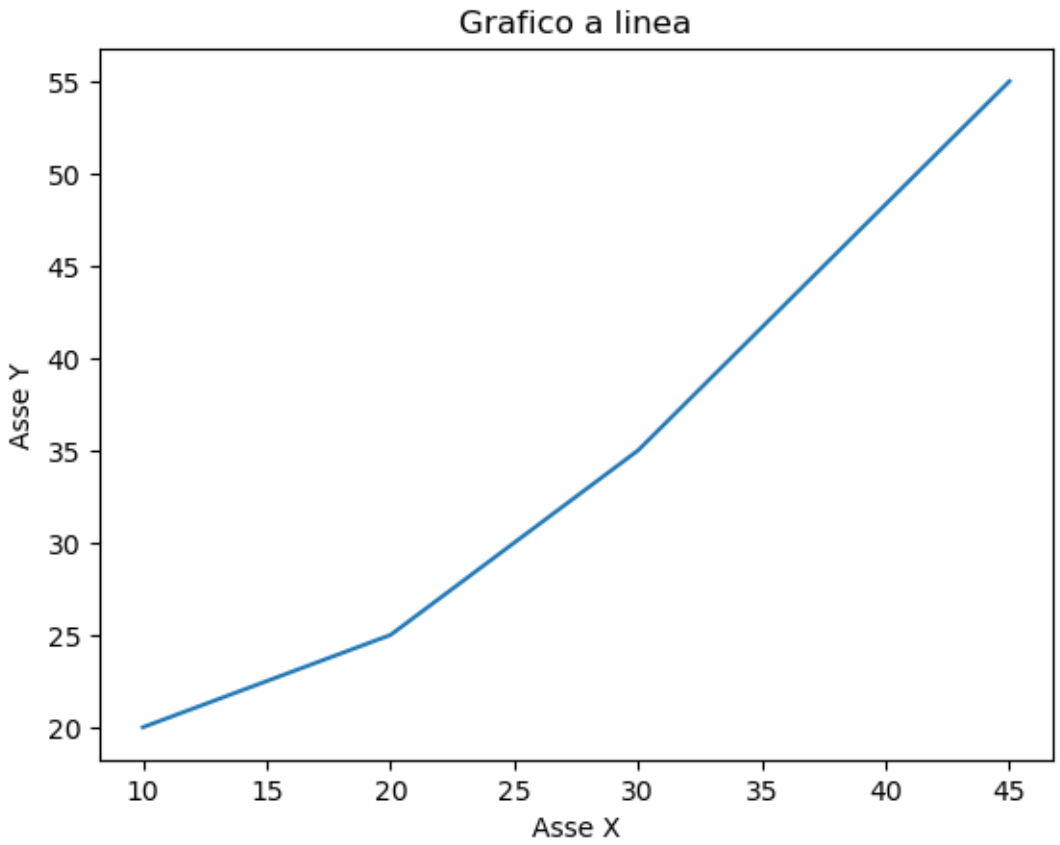
Il grafico a lina è uno dei grafici di base e può essere creato utilizzando la funzione ***plot()***. Viene utilizzato per rappresentare una relazione tra due dati X e Y su assi diversi.

Sintassi:

pyplot.plot(dati_x, dati_y)

La funzione prende in input i valori per i dati da mostrare sull'asse x e sull'asse y. Il modo più semplice per passare questi dati è sotto forma di iterabile (ad esempio una lista). Vediamo un esempio:

[24]:
```
from matplotlib import pyplot as plt

x = [10, 20, 30, 45]
y = [20, 25, 35, 55]

plt.plot(x, y)
plt.title('Grafico a linea')
plt.ylabel('Asse Y')
plt.xlabel('Asse X')
plt.show()
#plt.clf()
```



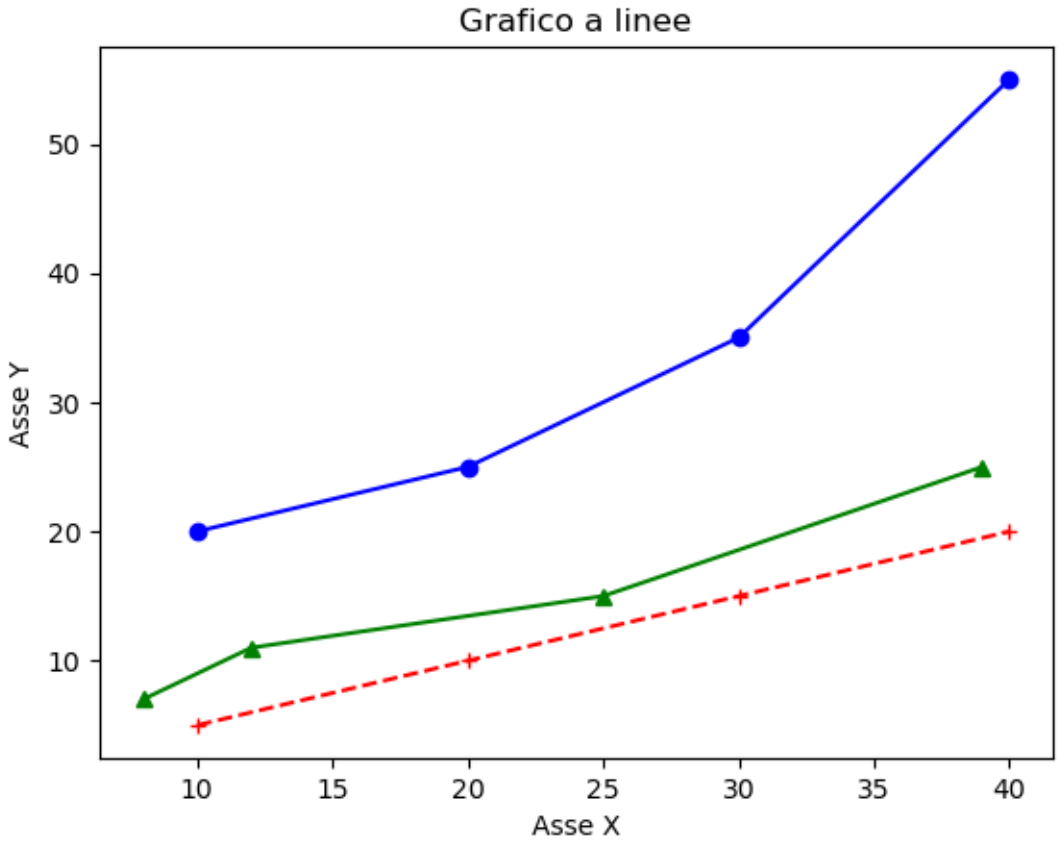
Nell'esempio che abbiamo appena visto sono presenti le funzioni più comuni di pyplot per produrre e mostrare un grafico. In generale, possiamo riassumere la generazione e formattazione di un grafico nei seguenti passaggi:

1. plt.nome_funzione(*argomento1*, *argomento2*, ...) - Chiamiamo una funzione di pyplot per generare il grafico di nostro interesse;
2. plt.title(*titolo*) - Questa funzione ci permette di aggiungere un titolo al grafico;
3. plt.ylabel(*label*) / plt.xlabel(*label*) - Possiamo specificare delle labels per contrassegnare i due assi del grafico;
4. plt.show() - visualizza il grafico che abbiamo preparato in una nuova finestra;
5. plt.clf() - cancella il grafico

Possiamo anche *plottare* più di una relazione, e poi mostrarle tutte e 3 nello stesso grafico come nell'esempio seguente:

[25]:
```
x1 = [10, 20, 30, 40]
y1 = [20, 25, 35, 55]

x2 = [10, 20, 30, 40]
y2 = [5, 10, 15, 20]
```

```
x3 = [8, 12, 25, 39]
y3 = [7, 11, 15, 25]

plt.plot(x1, y1, 'bo-')
plt.plot(x2, y2, 'r+--')
plt.plot(x3, y3, 'g^-')

plt.title('Grafico a linee')
plt.ylabel('Asse Y')
plt.xlabel('Asse X')

plt.show()
```



Nell'esempio precedente possiamo vedere anche come sia possibile passare una stringa come terzo argomento alla funzione *plot()*, dove in particolare il primo carattere indica il colore, il secondo la forma per l'indicazione posizionale del singolo datapoint, e il terzo lo stille della linea.

1.1.2 2. Grafico a barre

Il grafico a barre visualizza dati categorici utilizzando barre rettangolari la cui lunghezza è proporzionale ai valori che rappresentano. Può essere tracciato verticalmente o orizzontalmente per

confrontare diverse categorie.

Sintassi:

pyplot.bar(x, height)

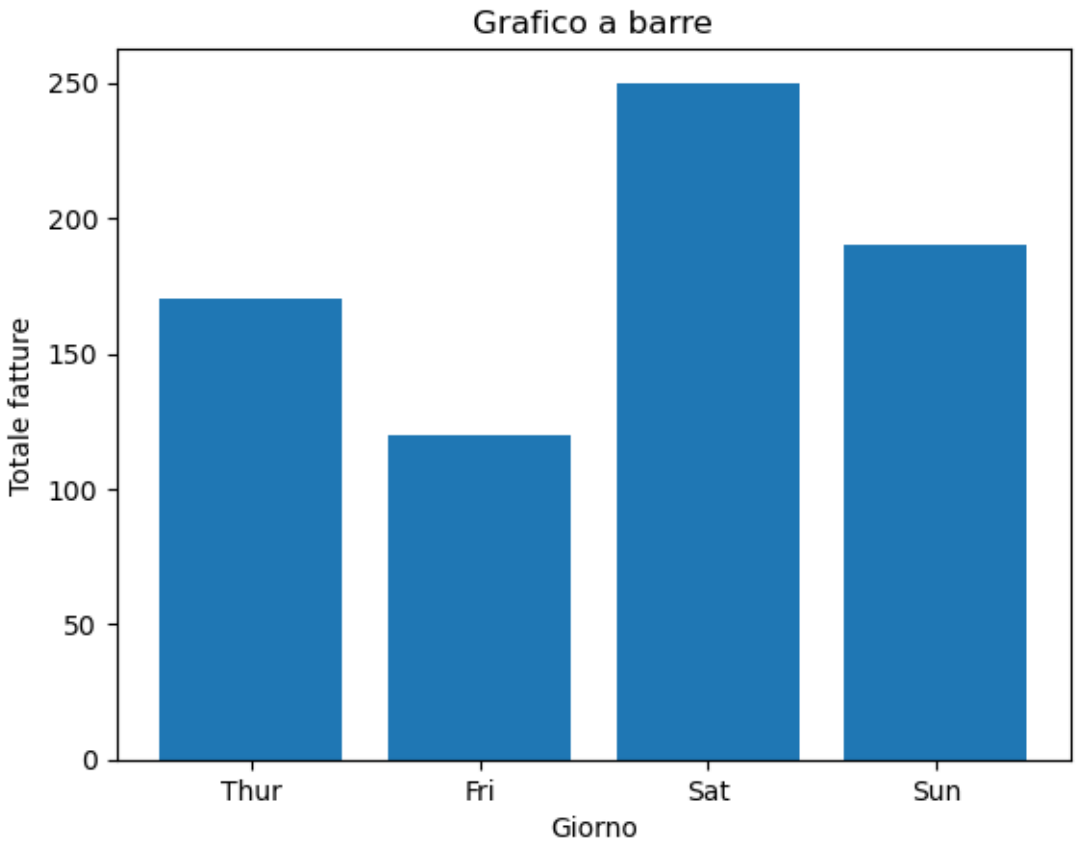
Dove:

x: Categorie o posizioni sull'asse x.

height: Altezza delle barre (valori dell'asse y).

Vediamo un semplice esempio:

[26]:
```
x = ['Thur', 'Fri', 'Sat', 'Sun']
y = [170, 120, 250, 190]

plt.bar(x, y)
plt.title('Grafico a barre')
plt.xlabel('Giorno')
plt.ylabel('Totale fatture')
plt.show()
plt.clf()
```



```
<Figure size 640x480 with 0 Axes>
```

1.1.3 3. Istogramma

L'istogramma mostra la distribuzione dei dati raggruppando i valori in intervalli. Per crearlo si utilizza la funzione ***hist()***, con l'asse X che mostra gli intervalli e l'asse Y che mostra le frequenze.

Sintassi:

pyplot.hist(x, bins=None)

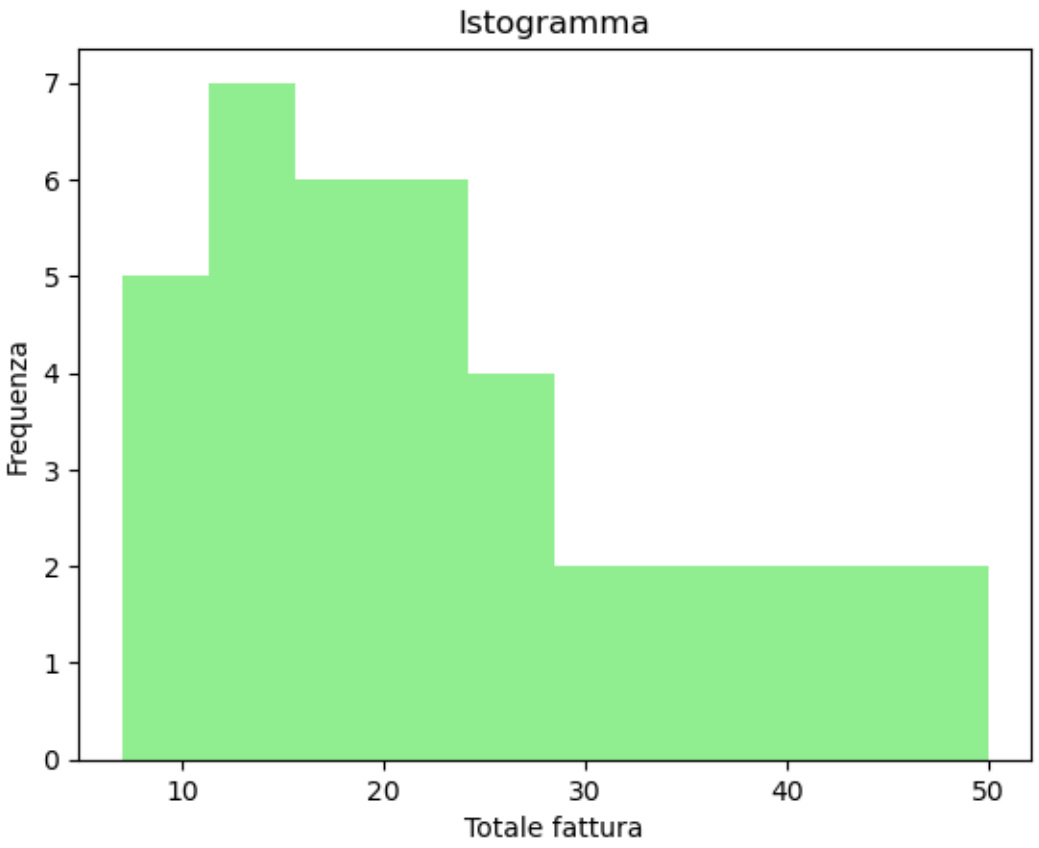
Dove:

x: dati di input.

bins: Numero di intervalli per raggruppare i dati.

Esempio: Questo codice traccia un istogramma per mostrare la distribuzione di frequenza dei valori totali delle fatture dall'elenco x. Utilizza 10 intervalli e aggiunge etichette agli assi e un titolo per maggiore chiarezza

[33]:
```
x = [7, 8, 9, 10, 10, 12, 12, 12, 13, 14, 14, 15, 16, 16, 17, 18, 18, 19, 20,␣
 ↪20,
     21, 22, 23, 24, 25, 25, 26, 28, 30, 32, 35, 36, 38, 40, 42, 44, 48, 50]

plt.hist(x, bins=10, color='lightgreen')
plt.title('Istogramma')
plt.xlabel('Totale fattura')
plt.ylabel('Frequenza')
plt.show()
```



1.1.4 4. Scatterplot

Gli scatterplot (che protremmo tradurre come *grafico di dispersione*) vengono utilizzati per osservare le relazioni tra le variabili. Detto in parole semplici, utilizzano i valori di due attributi per lo stesso record come coordinate X,Y di un piano cartesiano per disegnare dei punti. Per produrne uno in Python possiamo utilizzare il metodo ***scatter()*** del modulo pyplot.

Sintassi:

pyplot.scatter(x, y)

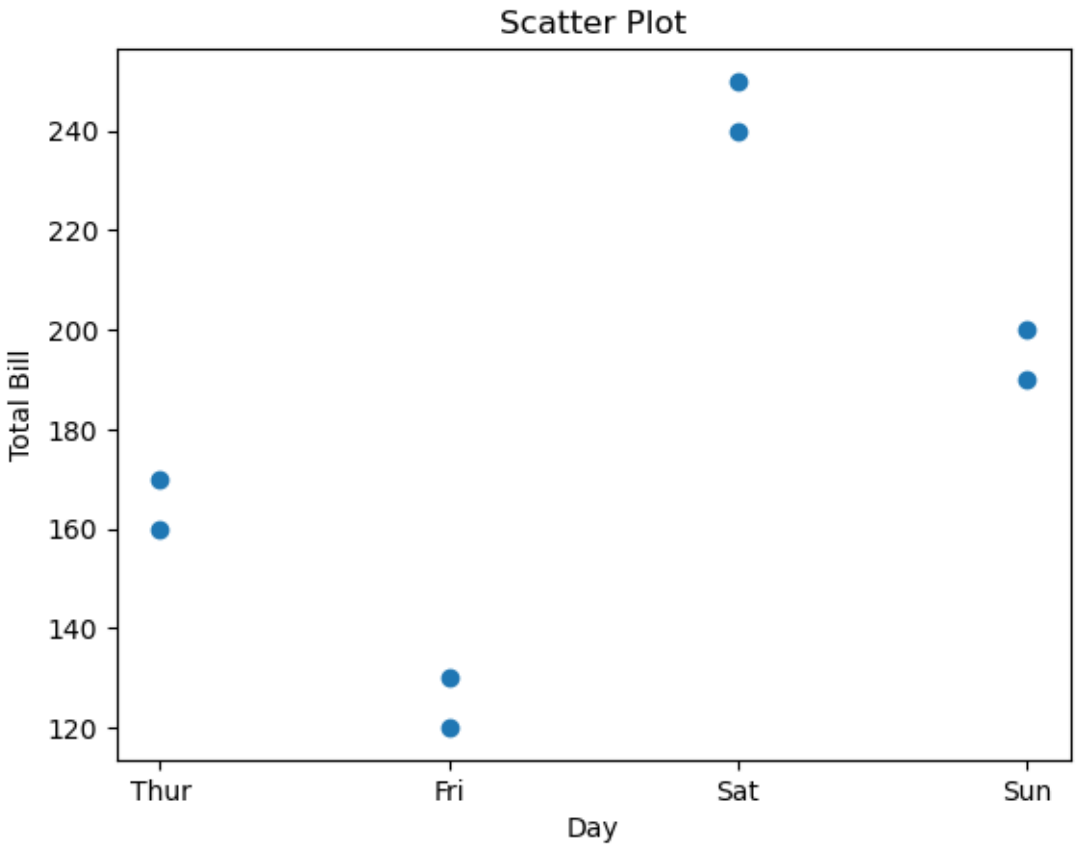
Dove:

x, *y* sono rispettivamente le coordinate del punto sull'asse delle ordinate e delle ascisse.

Nell'esempio seguente creiamo un grafico a dispersione per visualizzare la relazione tra i giorni della settimana e gli importi totali delle fatture:

[56]:
```
x = ['Thur', 'Fri', 'Sat', 'Sun', 'Thur', 'Fri', 'Sat', 'Sun']
y = [170, 120, 250, 190, 160, 130, 240, 200]

plt.scatter(x, y)
plt.title("Scatter Plot")
```

```
plt.xlabel("Day")
plt.ylabel("Total Bill")
plt.show()
```



Con i dati che abbiamo utilizzato sopra, l'utilizzo dello scatterplot potrebbe sembrare poco utile e/o informativo.

Tuttavia vediamo cosa succede nell'esempio successivo. Quello che faremo sarà utilizzare **NumPy** (libreria Python estremamente potente per lavorare con i numeri) per generare una serie di 200 numeri. Questi andranno da 1 a 100 e saranno crescenti in modo costante (tra tutti i numeri ci sarà la stessa distanza, quindi con un incremento di 0,5 tra ciascun numero).

Queste saranno le mie coordinate per l'asse X.

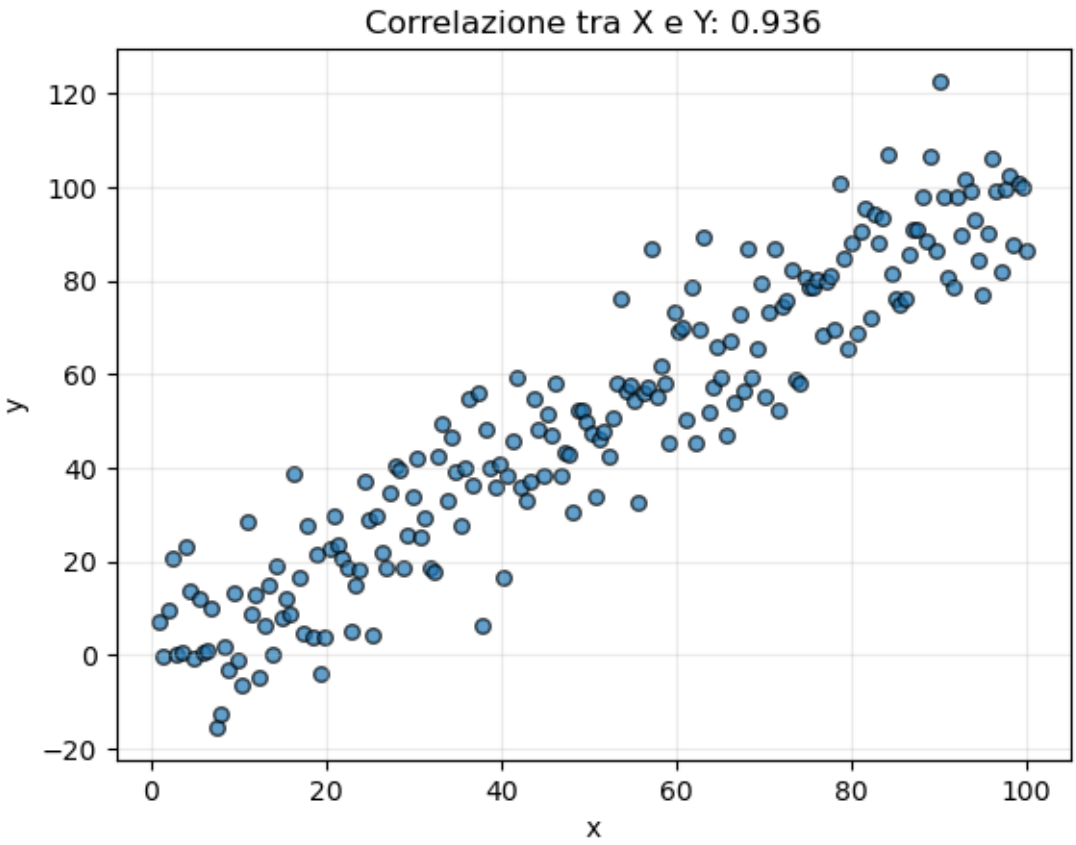
Successivamente andiamo a generare anche le coordinate per l'asse Y. Vogliamo che i dati mostrino una *correlazione lineare positiva*. Questo significa che all'aumentare di X dovrà aumentare in modo proporzionale anche Y. Se X e Y fossero uguali otterremo una linea diagonale perfetta, tuttavia vogliamo qualcosa che somigli un po' di più ad un caso reale, quindi dobbiamo creare un po' di confusione.

Definiamo quindi Y come X + *rumore*, dove rumore sarà un numero casuale campionato da una distribuzione normale con media 0 e varianza 12.

Anche per generare il *rumore* possiamo usare numpy, con la funzione *normal()* del modulo *numpy.random*

[74]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 200

#Utilizzo np.linspace() per generare 200 numeri equidistanti tra loro. Gli␣
 ↪argomenti sono:
# 1 - primo numero da generare
# 100 - la cifra a cui dobbiamo arrivare
# n = 200 - la quantità di numeri da generare
x = np.linspace(1, 100, n)

#Creo 200 numeri casuali "campionati" (ovvero che seguono) una distribuzione␣
 ↪normale
#np.random.normal() fa proprio questo prendendo come argomenti:
# loc - la media
# scale - la deviazione standard
# size - la quantità di numeri casuali da generare
#quindi avrò come risultato 200 numeri campionati da una distribuzione normale␣
 ↪con media 0 e deviazione standard 12
rumore = np.random.normal(loc=0, scale=12, size=n)

# Definisco y come x + rumore
y = (x + rumore)

# Calcolo la correlazione di pearson tra i due elementi
r = np.corrcoef(x, y)[0, 1]
print(f"Correlazione (r): {r:.3f}")

# Scatterplot
plt.scatter(x, y, alpha=0.7, edgecolor='k', s=30)
plt.title(f"Correlazione tra X e Y: {r:.3f}")
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, alpha=0.25)
plt.show()
```

```
Correlazione (r): 0.936
```



Per calcolare la correlazione tra le due variabili X ed Y utilizziamo il *metodo di Pearson*. Senza scendere nel dettaglio, questo metodo ci restituisce un numero compreso tra -1 ed 1.

-1 indica una correlazione negativa perfetta (all'aumentare di X, Y diminuisce nello stesso identico modo e viceversa). Ad esempio, se X aumenta di 2, Y diminuisce di 2 e così via.

1 al contrario, una forte correlazione positiva (all'aumentare di X, Y aumenta nello stesso identico modo).

Nella realtà è praticamente impossibile incontrare un coefficiente di 1 o -1 tra due variabili, ma valori intermedi possono comunque farci capire se esiste una correlazione positiva o negativa e quanto questa sia forte.

1.1.5 5. Grafico a torta

Il grafico a torta è un grafico circolare utilizzato per mostrare i dati sotto forma di proporzioni o percentuali. Viene creato utilizzando la funzione **pie()**, dove ogni fetta (sezione) rappresenta una parte del tutto.

Sintassi:

pyplot.pie(x, labels=None, autopct=None)

Dove:

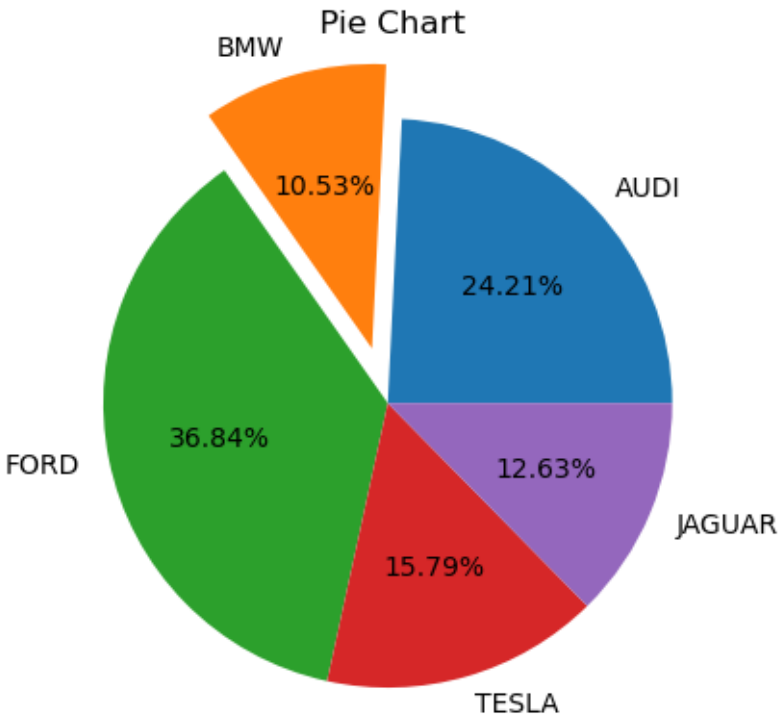
x: valori dei dati per le fette del grafico a torta.

labels: nomi per ciascuna fetta.

autopct: formato per visualizzare la percentuale (ad esempio, '%.2f%%').

Creiamo un semplice grafico a torta per visualizzare le vendite delle diverse marche di automobili in un concessionario. Ciascuna fetta della torta rappresenta la proporzione di automobili per ciascuna marca

[95]:
```
cars = ['AUDI', 'BMW', 'FORD','TESLA', 'JAGUAR',]
data = [23, 10, 35, 15, 12]
explode = [0.0, 0.2, 0, 0, 0]

plt.pie(data, labels=cars, autopct='%.2f%%', explode=explode)
plt.title(" Pie Chart")
plt.show()
```



1.1.6 6. Box Plot

Il box plot è un grafico semplice che mostra la distribuzione dei dati. Visualizza il minimo, il massimo, la mediana e i quartili e aiuta anche a individuare facilmente i valori anomali.

Sintassi:

pyplot.boxplot(x, notch=False, vert=True)

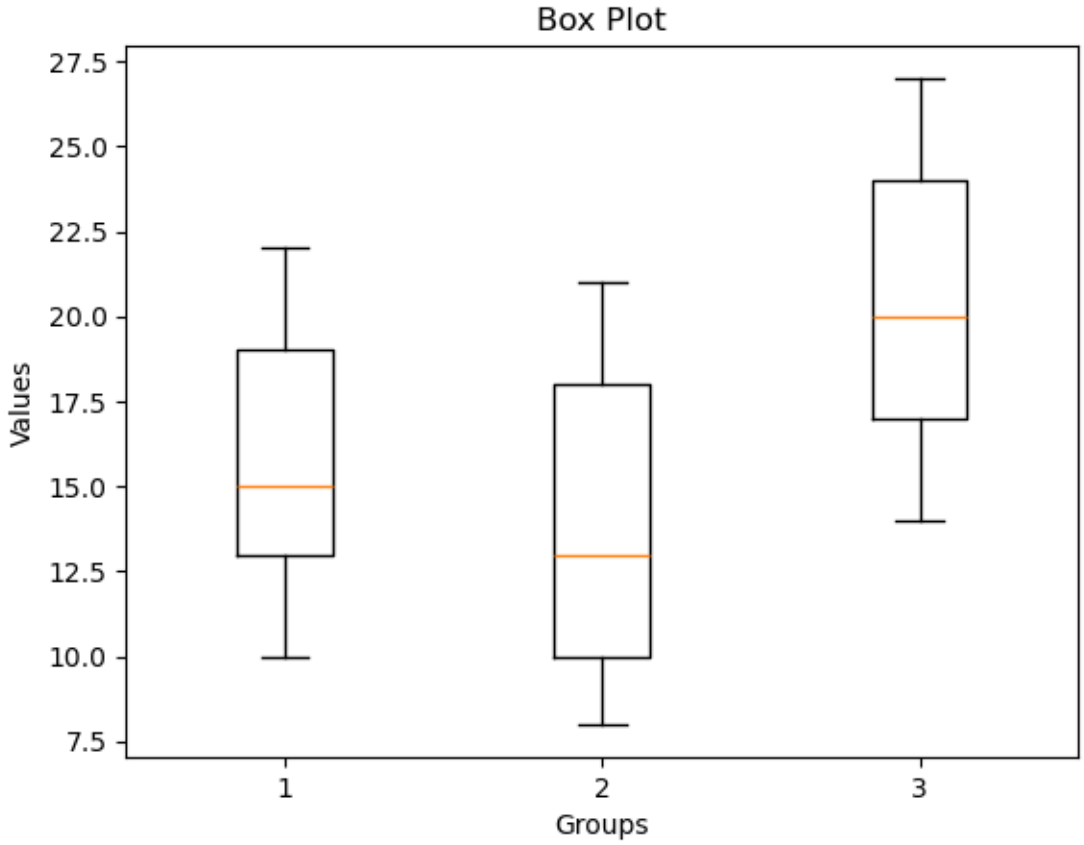
Parametri:

x: dati per i quali deve essere disegnato il box plot (di solito un elenco o un array).

notch: se True, disegna una tacca per mostrare l'intervallo di confidenza intorno alla mediana.

vert: se True, i box sono verticali. Se False, sono orizzontali.

[86]:
```
data = [ [10, 12, 14, 15, 18, 20, 22],
         [8, 9, 11, 13, 17, 19, 21],
         [14, 16, 18, 20, 23, 25, 27] ]

plt.boxplot(data, vert=True, notch=False)
plt.xlabel("Groups")
plt.ylabel("Values")
plt.title("Box Plot")
plt.show()
```



1.1.7 7. Heatmap

La Heatmap è una rappresentazione grafica dei dati in cui i valori sono mostrati come colori. Aiuta a visualizzare modelli, correlazioni o intensità in un formato simile a una matrice. Viene creata utilizzando il metodo ***imshow()***

Sintassi:

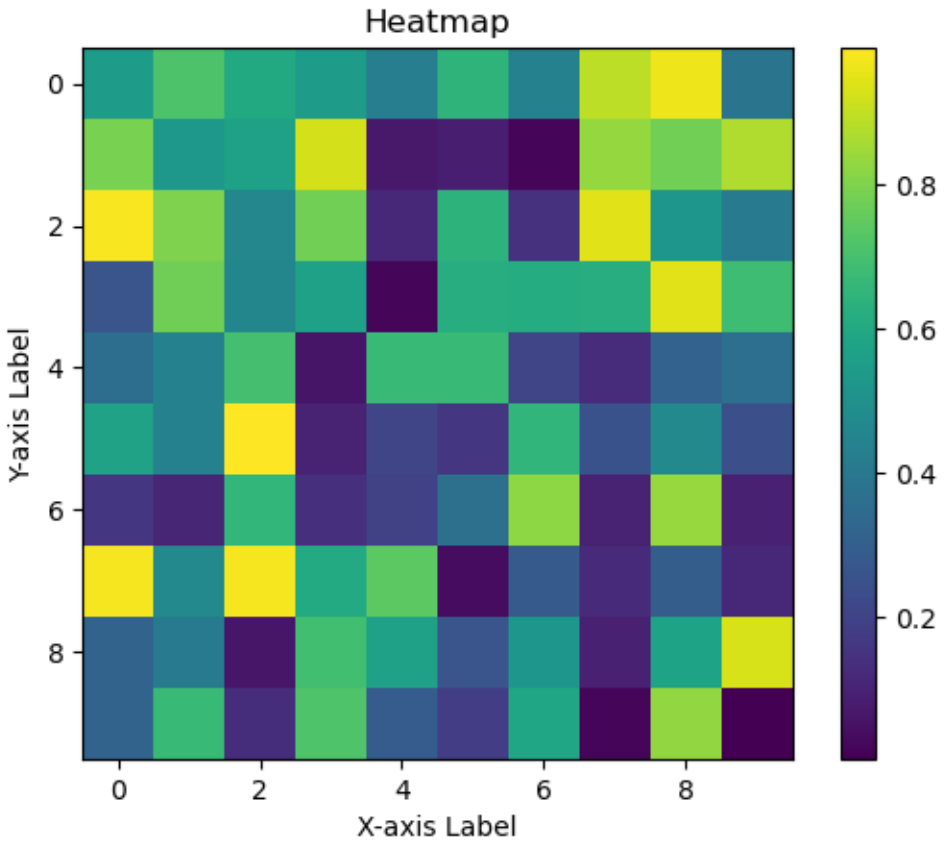
pyplot.imshow(X, cmap='viridis')

Parametri:

X: matrice 2x2 (dati da visualizzare)

cmap: imposta la mappa dei colori

[97]:
```
np.random.seed(0)
data = np.random.rand(10, 10)

plt.imshow(data, cmap='viridis')

plt.colorbar()
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Heatmap')
plt.show()
```



[ ]: