

# Il Primo Programma

Iniziamo scrivendo una frase tipica per iniziare un esercitazione python

```
In [1]: print("Ciao, mondo!")
```

Ciao, mondo!

```
In [ ]: nome="mattia"  
print(nome)
```

## Variabili e Input Utente

Iniziamo utilizzando le variabili

```
In [5]: nome = input("Inserisci il tuo nome: ")  
print("Ciao,", nome, "!"")
```

Inserisci il tuo nome: jfksjfkjskfjdeògfjeljgljle  
Ciao, jfksjfkjskfjdeògfjeljgljle !

```
In [ ]:
```

```
In [10]: via=input("inserisci nome della via")  
print("hai inserito",via)
```

inserisci nome della viaGluck  
hai inserito Gluck

Creiamo una ripetizione

```
In [11]: nome = input("Inserisci il tuo nome: ")  
for contatore in range(10):  
    print("Ciao,", nome, "!"") #Python ripete solo la parte di codice indentata
```

Inserisci il tuo nome: m  
Ciao, m !  
Ciao, m !

# Calcolatrice Python

## Addizione

```
In [1]: numero1 = int(input("Inserisci il primo numero: "))
numero2 = int(input("Inserisci il secondo numero: "))
somma = numero1 + numero2
print("La somma è:", somma)
```

```
Inserisci il primo numero: 4
Inserisci il secondo numero: 5
La somma è: 9
```

## Sottrazione

```
In [2]: sottrazione = numero1 - numero2
print("La sottrazione è:", sottrazione)
```

```
La sottrazione è: -1
```

## Moltiplicazione

```
In [1]: numero1 = int(input("Inserisci il primo numero: "))
numero2 = int(input("Inserisci il secondo numero: "))
moltiplicazione = numero1 * numero2
print("La moltiplicazione è:", moltiplicazione)
```

```
Inserisci il primo numero: 4
Inserisci il secondo numero: 6
La moltiplicazione è: 24
```

## Divisione

```
In [2]: divisione = numero1 / numero2
print("La divisione è:", divisione)
```

```
La divisione è: 0.6666666666666666
```

## Loop e Ripetizione

Stampare i primi 10 numeri interi positivi

```
In [5]: for numero in range(1,10+1):
    print(numero)
```

```
1
2
3
4
5
6
7
8
9
10
```

## Condizioni e Decisioni

Aggiungiamo delle decisioni

```
In [8]: # Calcolatrice Python con decisioni

operazione = input("Inserisci l'operazione (+, -, *, /): ")

numero1 = float(input("Inserisci il primo numero: "))
numero2 = float(input("Inserisci il secondo numero: "))

if operazione == "+":
    risultato = numero1 + numero2
elif operazione == "-":
    risultato = numero1 - numero2
elif operazione == "*":
    risultato = numero1 * numero2
elif operazione == "/":
    risultato = numero1 / numero2
else:
    risultato = "Operazione non valida"

print("Il risultato è:", risultato)
```

```
Inserisci l'operazione (+, -, *, /): %
Inserisci il primo numero: 3
Inserisci il secondo numero: 4
Il risultato è: Operazione non valida
```

## Contare fino a N

Ora faremo un loop

```
In [10]: n = int(input("Inserisci un numero intero positivo:"))
```

```
for numero in range(5,n+1):
    print(numero)
```

```
Inserisci un numero intero positivo:18
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

## Calcolare la Somma

Ora creeremo un calcolatrice con python

```
In [13]: n = int(input("Inserisci un numero intero positivo: "))
somma = 0
```

```
for numero in range(1, n+1):
    #somma = somma+ numero
    somma += numero
print("La somma dei primi", n, "numeri interi è:", somma)
```

```
Inserisci un numero intero positivo: 5
La somma dei primi 5 numeri interi è: 15
```

## Calcolare il Quadrato dei Primi Numeri

```
In [16]: n = int(input("Inserisci un numero intero positivo: "))
```

```
print("Quadrati dei primi", n, "numeri:")
for numero in range(1,n+1):
    quadrato = numero ** 2
    print("Il quadrato di", numero, "è", quadrato)
```

```
Inserisci un numero intero positivo: 5
Quadrati dei primi 5 numeri:
Il quadrato di 1 è 1
Il quadrato di 2 è 4
Il quadrato di 3 è 9
Il quadrato di 4 è 16
Il quadrato di 5 è 25
```

# Verificare la Parità

Vediamo come si comporta con delle condizioni

```
In [6]: numero = int(input("Inserisci un numero: "))

if numero % 2 == 0:
    print(numero, "è un numero pari.")
else:
    print(numero, "è un numero dispari.)
```

```
Inserisci un numero: 5
5 è un numero dispari.
```

# Calcolare il Fattoriale

```
In [8]: n = int(input("Inserisci un numero intero positivo: "))
fattoriale = 1

for numero in range(1, n + 1):
    fattoriale = fattoriale*numero
    #fattoriale *=numero
print("Il fattoriale di", n, "è:", fattoriale)
```

```
Inserisci un numero intero positivo: 5
Il fattoriale di 5 è: 120
```

# Calcolare la Media di una Lista di Numeri

Proviamo a calcolare la media

```
In [11]: numeri = []

n = int(input("Quanti numeri vuoi inserire? "))

for i in range(n):
    numero = float(input("Inserisci un numero: "))
    numeri.append(numero)

media = sum(numeri) / len(numeri)

print("La media dei numeri inseriti è:", media,"la lista completa è:", numeri)
```

```
Quanti numeri vuoi inserire? 6
Inserisci un numero: 3
Inserisci un numero: 4
Inserisci un numero: 5
Inserisci un numero: 6
Inserisci un numero: 7
Inserisci un numero: 7
La media dei numeri inseriti è: 5.333333333333333 la lista completa è: [3.0, 4.0, 5.0, 6.0, 7.0, 7.0]
```

## Gioco dell'Indovinello

```
In [1]: import random

numero_da_indotinare = random.randint(1, 100)
tentativi = 0

while True:
    tentativo = int(input("Indovina il numero (1-100): "))
    tentativi += 1

    if tentativo == numero_da_indotinare:
        print("Bravo! Hai indovinato il numero", numero_da_indotinare, "in")
        break
    elif tentativo < numero_da_indotinare:
        print("Il numero da indovinare è più grande.")
    else:
        print("Il numero da indovinare è più piccolo.)
```

```
Indovina il numero (1-100): 10
Il numero da indovinare è più grande.
Indovina il numero (1-100): 90
Il numero da indovinare è più piccolo.
Indovina il numero (1-100): 20
Il numero da indovinare è più grande.
Indovina il numero (1-100): 80
Il numero da indovinare è più piccolo.
Indovina il numero (1-100): 40
Il numero da indovinare è più piccolo.
Indovina il numero (1-100): 30
Il numero da indovinare è più grande.
Indovina il numero (1-100): 35
Il numero da indovinare è più piccolo.
Indovina il numero (1-100): 32
Il numero da indovinare è più piccolo.
Indovina il numero (1-100): 31
Bravo! Hai indovinato il numero 31 in 9 tentativi.
```

## Gioco del Morra Cinese

```
In [2]: import random

mosse = ["carta", "forbici", "sasso"]

computer_mossa = random.choice(mosse)

print("Benvenuti al Gioco del Morra Cinese!")
scelta_giocatore = input("Scegli la tua mossa (carta, forbici, sasso): ")

if scelta_giocatore not in mosse:
    print("mossa non permessa")
else:
    print("Il computer ha scelto:", computer_mossa)
    if scelta_giocatore == computer_mossa:
        print("Pareggio!")
    elif (scelta_giocatore == "carta" and computer_mossa == "sasso") or \
        (scelta_giocatore == "forbici" and computer_mossa == "carta") or \
        (scelta_giocatore == "sasso" and computer_mossa == "forbici"):
        print("Hai vinto!")
    else:
        print("Hai perso!")
```

Benvenuti al Gioco del Morra Cinese!  
Scegli la tua mossa (carta, forbici, sasso): carta  
Il computer ha scelto: forbici  
Hai perso!

In [ ]:

In [ ]:

In [ ]:

In [2]:

```
n=int(input("inserisci un numero intero: "))

fattoriale=1

if n<0:
    print("numero negativo")
elif n==0:
    print("il fattoriale di zero è 1 per definizione")
else:
    for numero in range(1,n+1):
        fattoriale*=numero
print(f"il fattoriale di {n} è {fattoriale}")
```

inserisci un numero intero: 4  
il fattoriale di 4 è 24

In [3]:

```
# Chiedere all'utente di inserire un numero intero positivo N
N = int(input("Inserisci un numero intero positivo N: "))

# Inizializzare La somma a zero
somma = 0

# Calcolare La somma dei primi N numeri pari
for numero in range(2, 2 * N + 1, 2):
    somma += numero
print(f"La somma dei primi {N} numeri pari è {somma}")
```

Inserisci un numero intero positivo N: 5  
La somma dei primi 5 numeri pari è 30

In [4]:

```
# Chiedere all'utente di inserire un numero intero positivo N
N = int(input("Inserisci un numero intero positivo N: "))
lista=[]

# Calcolare La somma dei primi N numeri pari
for numero in range(2, 2 * N + 1, 2):
    lista.append(numero)

print(lista)
```

Inserisci un numero intero positivo N: 4  
[2, 4, 6, 8]

```
In [5]: # Chiedi all'utente di inserire una frase o una parola
frase = input("Inserisci una frase o una parola: ").lower() # Converti tutto in minuscolo

# Inizializza il contatore delle vocali
conteggio_vocali = 0

# Definisci le vocali da cercare
vocali = "aeiou"

# Scansiona ogni carattere nella frase
for carattere in frase:
    # Verifica se il carattere è una vocale
    if carattere in vocali:
        conteggio_vocali += 1

# Stampa il conteggio delle vocali
print(f"Nella frase inserita ci sono {conteggio_vocali} vocali.")
```

Inserisci una frase o una parola: Piero è andato via  
Nella frase inserita ci sono 8 vocali.

```
In [ ]:
```

```
In [6]: import random
```

```
# Genera un numero casuale da 1 a 6 (simulando il lancio di un dado)
numero_dado = random.randint(1, 6)

# Chiedi all'utente di indovinare il numero
indovina = int(input("Indovina il numero del dado (da 1 a 6): "))

# Verifica se l'utente ha indovinato correttamente
if indovina <1 or indovina >6:
    print("numero non ammesso")
elif indovina == numero_dado:
    print(f"Complimenti! Il numero del dado era {numero_dado}. Hai indovinato!")
else:
    print(f"Mi dispiace, il numero del dado era {numero_dado}. Meglio fortuna la prossima!")
```

Indovina il numero del dado (da 1 a 6): 4  
Mi dispiace, il numero del dado era 1. Meglio fortuna alla prossima!

```
In [7]: # Inizializza la popolazione e gli anni
popolazione = int(input("inserisce popolazione iniziale: "))
anni = int(input("inserisci numero di anni da simulare: "))
# Tasso di natalità e tasso di mortalità (percentuale annuale)
tasso_natalita = float(input("inserisci tasso natalità: "))
tasso_mortalita = float(input("inserisci tasso mortalità: "))

# Simulazione della crescita della popolazione
for anno in range(anni):
    nascite = (popolazione * tasso_natalita) / 100
    morti = (popolazione * tasso_mortalita) / 100
    popolazione += (nascite - morti)

    print(f"Anno {anno+1}: Popolazione = {int(popolazione)}")

print("Simulazione completata.")
```

```
inserisce popolazione iniziale: 10000
inserisci numero di anni da simulare: 10
inserisci tasso natalità: 3.3
inserisci tasso mortalità: 1.5
Anno 1: Popolazione = 10180
Anno 2: Popolazione = 10363
Anno 3: Popolazione = 10549
Anno 4: Popolazione = 10739
Anno 5: Popolazione = 10932
Anno 6: Popolazione = 11129
Anno 7: Popolazione = 11330
Anno 8: Popolazione = 11534
Anno 9: Popolazione = 11741
Anno 10: Popolazione = 11953
Simulazione completata.
```

```
In [9]: # import math

print("Benvenuto nel Risolutore di Equazioni di Secondo Grado!")
print("L'equazione deve essere nella forma ax^2 + bx + c = 0")

# Chiedi all'utente di inserire i coefficienti
a = float(input("Inserisci il coefficiente 'a': "))
b = float(input("Inserisci il coefficiente 'b': "))
c = float(input("Inserisci il coefficiente 'c': "))

# Calcola il discriminante
discriminante = b**2 - 4*a*c

# Verifica se l'equazione ha soluzioni reali
if discriminante > 0:
    soluzione1 = (-b + math.sqrt(discriminante)) / (2*a)
    soluzione2 = (-b - math.sqrt(discriminante)) / (2*a)
    print(f'L'equazione ha due soluzioni reali: x1 = {soluzione1:.2f} e x2 = {soluzione2:.2f}')
elif discriminante == 0:
    soluzione = -b / (2*a)
    print(f'L'equazione ha una soluzione reale doppia: x = {soluzione:.2f}')
else:
    parte_reale = -b / (2*a)
    parte_immaginaria = math.sqrt(-discriminante) / (2*a)
    print(f'L'equazione ha due soluzioni complesse: x1 = {parte_reale:.2f} + {parte_immaginaria:.2f}i e x2 = {parte_reale:.2f} - {parte_immaginaria:.2f}i')
```

```
Benvenuto nel Risolutore di Equazioni di Secondo Grado!
L'equazione deve essere nella forma ax^2 + bx + c = 0
Inserisci il coefficiente 'a': 1
Inserisci il coefficiente 'b': 0
Inserisci il coefficiente 'c': 0
L'equazione ha una soluzione reale doppia: x = -0.00
```

```
In [10]: import datetime

today = datetime.datetime.today()
print(f"oggi è il giorno: {today:%d %m %Y} ore: {today:%H %M %S}")
```

```
oggi è il giorno: 26 10 2023 ore: 02 04 57
```

```
In [11]: print("Benvenuto nel Convertitore di Unità di Misura!")
scelta = input("Cosa desideri convertire? (metri/piedi/chilogrammi/libbre):")

if scelta == "metri":
    valore = float(input("Inserisci il valore in metri: "))
    risultato = valore * 3.28084
    print(f"{valore} metri corrispondono a {risultato} piedi.")

elif scelta == "piedi":
    valore = float(input("Inserisci il valore in piedi: "))
    risultato = valore / 3.28084
    print(f"{valore} piedi corrispondono a {risultato} metri.")
elif scelta == "chilogrammi":
    valore = float(input("Inserisci il valore in chilogrammi: "))
    risultato = valore * 2.20462
    print(f"{valore} chilogrammi corrispondono a {risultato} libbre.")
elif scelta == "libbre":
    valore = float(input("Inserisci il valore in libbre: "))
    risultato = valore / 2.20462
    print(f"{valore} libbre corrispondono a {risultato} chilogrammi.")
else:
    print("Scelta non valida. Scegli tra 'metri', 'piedi', 'chilogrammi' o")
```

Benvenuto nel Convertitore di Unità di Misura!  
Cosa desideri convertire? (metri/piedi/chilogrammi/libbre): metri  
Inserisci il valore in metri: 100  
100.0 metri corrispondono a 328.084 piedi.

In [ ]:

```
In [12]: # Chiedere all'utente di inserire un numero n
n = int(input("Inserisci un numero n per calcolare l'n-esimo numero di Fibonacci"))
# Inizializzare le variabili per i primi due numeri di Fibonacci
a=0
b=1
c=a+b

# Calcolare L'n-esimo numero di Fibonacci
if n <= 0:
    print("Il numero deve essere maggiore di zero.")
elif n == 1:
    risultato = a
else:
    for iterazione in range(n-3):
        a = b
        b = c
        c = a + b
    risultato = c
# Stampare L'n-esimo numero di Fibonacci
print("L'n-esimo numero di Fibonacci è:", risultato)
```

Inserisci un numero n per calcolare l'n-esimo numero di Fibonacci: 4  
L'n-esimo numero di Fibonacci è: 2

## FUNZIONI CUSTOM

```
In [13]: def fibonacci(n):
    fib_series = [0, 1]

    while len(fib_series) < n:
        fib_series.append(fib_series[-1] + fib_series[-2])

    return fib_series
```

```
In [ ]:
```

```
In [14]: fibonacci(15)
```

```
Out[14]: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

```
In [15]: n = int(input("Inserisci il numero di termini della serie di Fibonacci da generare:"))

if n <= 0:
    print("Inserisci un numero positivo.")
else:
    result = fibonacci(n)
    print(result)
```

Inserisci il numero di termini della serie di Fibonacci da generare: 30  
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229]

```
In [16]: import math

def calcola_area_cerchio(raggio):
    return math.pi * (raggio ** 2)

def calcola_area_rettangolo(base, altezza):
    return base * altezza

def calcola_area_triangolo(base, altezza):
    return (base * altezza) / 2
```

```
In [17]: calcola_area_cerchio(10)
```

```
Out[17]: 314.1592653589793
```

```
In [18]: print("Benvenuto nella Calcolatrice di Aree!")

scelta = input("Vuoi calcolare l'area di un cerchio (c), rettangolo (r) o triangolo (t)? ")

if scelta == 'c':
    raggio = float(input("Inserisci il raggio del cerchio: "))
    area = calcola_area_cerchio(raggio)
    print(f"L'area del cerchio è {area:.2f}")
elif scelta == 'r':
    base = float(input("Inserisci la base del rettangolo: "))
    altezza = float(input("Inserisci l'altezza del rettangolo: "))
    area = calcola_area_rettangolo(base, altezza)
    print(f"L'area del rettangolo è {area:.2f}")
elif scelta == 't':
    base = float(input("Inserisci la base del triangolo: "))
    altezza = float(input("Inserisci l'altezza del triangolo: "))
    area = calcola_area_triangolo(base, altezza)
    print(f"L'area del triangolo è {area:.2f}")
else:
    print("Scelta non valida. Si prega di inserire 'c', 'r' o 't'.")
```

Benvenuto nella Calcolatrice di Aree!  
 Vuoi calcolare l'area di un cerchio (c), rettangolo (r) o triangolo (t)? c  
 Inserisci il raggio del cerchio: 10  
 L'area del cerchio è 314.16

```
In [19]: def calcola_interessi(importo_iniziale, tasso_interesse, periodi_investimento):
    importo_finale = importo_iniziale * (1 + tasso_interesse / 100) ** periodi_investimento
    return importo_finale
```

```
In [20]: print("Benvenuto nel Calcolatore di Interessi!")

importo = float(input("Inserisci l'importo iniziale: "))
tasso = float(input("Inserisci il tasso di interesse annuale (%): "))
periodo = int(input("Inserisci il periodo di investimento (anni): "))

importo_finale = calcola_interessi(importo, tasso, periodo)

print(f"L'importo finale dopo {periodo} anni è di {importo_finale:.2f} euro.")
```

Benvenuto nel Calcolatore di Interessi!  
 Inserisci l'importo iniziale: 100000  
 Inserisci il tasso di interesse annuale (%): 2  
 Inserisci il periodo di investimento (anni): 10  
 L'importo finale dopo 10 anni è di 121899.44 euro.

```
In [21]: calcola_interessi(10000000,4,10)
```

Out[21]: 14802442.849183444

In [ ]:

```
In [22]: def forza_gravitazionale(m1, m2, r):
    # Costante gravitazionale
    G = 6.67430e-11 # N(m/kg)^2

    # Calcolo della forza gravitazionale
    F = (G * m1 * m2) / (r ** 2)

    return F
```

```
In [23]: # Esempio di utilizzo
massa_terra = 5.972e24 # kg
massa_luna = 7.342e22 # kg
distanza_terra_luna = 384400000 # metri

forza = forza_gravitazionale(massa_terra, massa_luna, distanza_terra_luna)
print(f"Forza gravitazionale tra la Terra e la Luna: {forza} Newton")
```

Forza gravitazionale tra la Terra e la Luna: 1.9804922390990566e+20 Newton

```
In [ ]:
```

```
In [29]: from itertools import permutations
k=0

def trova_anagrammi(parola):
    anagrammi = [".".join(p) for p in permutations(parola)]
    return anagrammi
print("Benvenuto nel Risolutore di Anagrammi!")

parola_input = input("Inserisci una parola: ").strip().lower()

if len(parola_input) < 2:
    print("Inserisci una parola con almeno 2 caratteri.")
else:
    anagrammi = trova_anagrammi(parola_input)

    for elemento in anagrammi:
        if elemento != parola_input:
            k+=1
            print(elemento)
    print(f"Gli anagrammi di '{parola_input}' sono: '{k}'")
```

Benvenuto nel Risolutore di Anagrammi!  
Inserisci una parola: Mat  
mta  
amt  
atm  
tma  
tam  
Gli anagrammi di 'mat' sono: '5'

In [ ]:

In [30]: #dizionari

In [31]: # Definizione dei tassi di cambio

```
tassi_di_cambio = {
    "dollari": 1.0,
    "euro": 0.85,
    "yen": 110.41,
    # Aggiungi altre valute e tassi di cambio se necessario
}

# Chiedi all'utente di inserire l'importo, la valuta di partenza e la valuta di destinazione
importo = float(input("Inserisci l'importo da convertire: "))
valuta_di_partenza = input("Inserisci la valuta di partenza: ").lower()
valuta_destinazione = input("Inserisci la valuta di destinazione: ").lower()

# Verifica se le valute sono nel dizionario dei tassi di cambio
if valuta_di_partenza in tassi_di_cambio and valuta_destinazione in tassi_di_cambio:
    # Calcola il tasso di cambio e l'importo convertito
    tasso_di_cambio = tassi_di_cambio[valuta_destinazione] / tassi_di_cambio[valuta_di_partenza]
    importo_convertito = importo * tasso_di_cambio

    # Stampa il risultato
    print(f"\n{importo} {valuta_di_partenza} sono equivalenti a {importo_convertito} {valuta_destinazione}")
else:
    print("Valute non supportate. Assicurati di inserire valute valide.")
```

```
Inserisci l'importo da convertire: 10000
Inserisci la valuta di partenza: euro
Inserisci la valuta di destinazione: yen
10000.0 euro sono equivalenti a 1298941.18 yen
```

In [32]: tassi\_di\_cambio["euro"]

Out[32]: 0.85

```
In [33]: # Chiedi all'utente di inserire una frase
frase = input("Inserisci una frase: ")

# Converti la frase in minuscolo per evitare problemi di maiuscole/minuscole
frase = frase.lower()

# Inizializza una lista di lettere dell'alfabeto
alfabeto = 'abcdefghijklmnopqrstuvwxyz'

# Inizializza un dizionario per tenere traccia del conteggio delle lettere
conteggio_lettere = {}

# Itera attraverso ciascuna lettera dell'alfabeto
for lettera in alfabeto:
    # Conta quante volte appare la lettera nella frase
    conteggio = frase.count(lettera)

    # Aggiungi la lettera e il conteggio al dizionario se la lettera appare
    if conteggio > 0:
        conteggio_lettere[lettera] = conteggio

# Stampa il conteggio delle lettere in un formato leggibile
for lettera, conteggio in conteggio_lettere.items():
    print(f"{lettera}: {conteggio}")
```

Inserisci una frase: ci sono tante persone fuori

```
a: 1
c: 1
e: 3
f: 1
i: 2
n: 3
o: 4
p: 1
r: 2
s: 2
t: 2
u: 1
```

```
In [34]: conteggio_lettere.items()
```

```
Out[34]: dict_items([('a', 1), ('c', 1), ('e', 3), ('f', 1), ('i', 2), ('n', 3), ('o', 4), ('p', 1), ('r', 2), ('s', 2), ('t', 2), ('u', 1)])
```

```
In [35]: prodotti={}
prodotti["pan bauletto"]=2
prodotti["coca cola"]=3
```

```
In [36]: prodottidue={
    "pan bauletto":2,
    "coca cola":3

}
```

```
In [37]: prodottidue
```

```
Out[37]: {'pan bauletto': 2, 'coca cola': 3}
```

```
In [38]: from datetime import datetime
import pytz

print("Benvenuto nell'Orologio Mondiale!")

# Definisci le città e i relativi fusi orari
citta_fusi_orari = {
    "New York": "America/New_York",
    "Londra": "Europe/London",
    "Tokyo": "Asia/Tokyo",
    "Sydney": "Australia/Sydney",
    "Rio de Janeiro": "America/Sao_Paulo",
}

while True:
    print("\nCittà disponibili:")
    for citta in citta_fusi_orari.keys():
        print(citta)

    scelta_citta = input("Inserisci il nome della città per visualizzare l'ora (o 'esci' per uscire): ")
    if scelta_citta.lower() == 'esci':
        break

    if scelta_citta in citta_fusi_orari.keys():
        fuso_orario = pytz.timezone(citta_fusi_orari[scelta_citta])
        ora_corrente = datetime.now(fuso_orario)
        print(f"L'ora corrente a {scelta_citta} è: {ora_corrente.strftime('%H:%M:%S')}")
    else:
        print("Città non valida. Riprova.")
```

Benvenuto nell'Orologio Mondiale!

Città disponibili:  
New York  
Londra  
Tokyo  
Sydney  
Rio de Janeiro  
Inserisci il nome della città per visualizzare l'ora (o 'esci' per uscire): New York  
L'ora corrente a New York è: 20:07:46

Città disponibili:  
New York  
Londra  
Tokyo  
Sydney  
Rio de Janeiro  
Inserisci il nome della città per visualizzare l'ora (o 'esci' per uscire): esci

In [ ]:

## dizionari e main

In [ ]:

In [ ]:

```
In [39]: # Funzione principale può avere qualsiasi nome
def paolo():
    print("mi chiamo paolo")

if __name__ == "__main__": #è una condizione logica che "si verifica sempre"
                           #che risulta indentato a questa condizione viene
                           paolo()

mi chiamo paolo
```

In [ ]:

```
In [40]: # Funzione principale può avere qualsiasi nome
def main():
    print("la funzione principale del codice è stata eseguita, in questa fur

if __name__ == "__main__":
    main()
```

la funzione principale del codice è stata eseguita, in questa funzione possono essere presenti funzioni secondarie precedentemente create

In [41]:

```
#main
# Funzione per il calcolo del BMI
def calcola_bmi(peso, altezza):
    return peso / (altezza ** 2)

# Funzione per la valutazione del BMI
def valuta_bmi(bmi):
    if bmi < 18.5:
        return "Sottopeso"
    elif 18.5 <= bmi < 24.9:
        return "Normopeso"
    elif 25 <= bmi < 29.9:
        return "Sovrappeso"
    else:
        return "Obeso"

# Funzione principale
def main():
    print("Benvenuto nella Calcolatrice BMI!")
    peso = float(input("Inserisci il tuo peso in chilogrammi: "))
    altezza = float(input("Inserisci la tua altezza in metri: "))
    bmi = calcola_bmi(peso, altezza)
    valutazione = valuta_bmi(bmi)
    print(f"Il tuo BMI è {bmi:.2f}, sei classificato come '{valutazione}'.")

if __name__ == "__main__":
    main()
```

Benvenuto nella Calcolatrice BMI!  
 Inserisci il tuo peso in chilogrammi: 90  
 Inserisci la tua altezza in metri: 1.90  
 Il tuo BMI è 24.93, sei classificato come 'Obeso'.

In [42]:

```
# Funzione principale
def main():
    n=int(input("inserisci numero di persone da valutare: "))
    for persone in range(n):
        print("Benvenuto nella Calcolatrice BMI!")
        peso = float(input("Inserisci il tuo peso in chilogrammi: "))
        altezza = float(input("Inserisci la tua altezza in metri: "))

        bmi = calcola_bmi(peso, altezza)
        valutazione = valuta_bmi(bmi)

        print(f"Il tuo BMI è {bmi:.2f}, sei classificato come '{valutazione}'")

    if __name__ == "__main__":
        main()
```

inserisci numero di persone da valutare: 2  
 Benvenuto nella Calcolatrice BMI!  
 Inserisci il tuo peso in chilogrammi: 80  
 Inserisci la tua altezza in metri: 1.90  
 Il tuo BMI è 22.16, sei classificato come 'Normopeso'.  
 Benvenuto nella Calcolatrice BMI!  
 Inserisci il tuo peso in chilogrammi: 72  
 Inserisci la tua altezza in metri: 1.84  
 Il tuo BMI è 21.27, sei classificato come 'Normopeso'.

```
In [43]: # Funzione per la conversione di metri in piedi
def metri_a_piedi(metri):
    return metri * 3.28084
def piedi_a_metri(piedi):
    return piedi / 3.28084
# Funzione per la conversione di chilogrammi in libbre
def chilogrammi_a_libbre(chilogrammi):
    return chilogrammi * 2.20462
def libbre_a_chilogrammi(libbre):
    return libbre / 2.20462

def selezione(scelta):
    if scelta == "metri":
        valore = float(input("Inserisci il valore in metri: "))
        risultato = metri_a_piedi(valore)
        print(f"{valore: .3f} metri corrispondono a {risultato: .3f} piedi.")
    elif scelta == "piedi":
        valore = float(input("Inserisci il valore in piedi: "))
        risultato = piedi_a_metri(valore)
        print(f"{valore: .3f} piedi corrispondono a {risultato: .3f} metri.")
    elif scelta == "chilogrammi":
        valore = float(input("Inserisci il valore in chilogrammi: "))
        risultato = chilogrammi_a_libbre(valore)
        print(f"{valore: .3f} chilogrammi corrispondono a {risultato: .3f} libbre.")
    elif scelta == "libbre":
        valore = float(input("Inserisci il valore in libbre: "))
        risultato = libbre_a_chilogrammi(valore)
        print(f"{valore: .3f} libbre corrispondono a {risultato: .3f} chilogrammi")
    else:
        print("Sveglia!!! Scelta non valida. Scegli tra 'metri', 'piedi', 'chilogrammi' o 'libbre'.")

# Funzione principale
def main():
    print("Benvenuto nel Convertitore di Unità di Misura!")
    scelta = input("Cosa desideri convertire? (metri/piedi/chilogrammi/libbre): ")
    selezione(scelta)
if __name__ == "__main__":
    main()
```

Benvenuto nel Convertitore di Unità di Misura!  
 Cosa desideri convertire? (metri/piedi/chilogrammi/libbre): metri  
 Inserisci il valore in metri: 100  
 100.000 metri corrispondono a 328.084 piedi.

In [ ]:

```
In [44]: # Dizionario per il cibo e le calorie per 100 grammi
cibo_calorie = {
    "pizza": 285,
    "hamburger": 250,
    "insalata": 100,
    "pollo arrosto": 335,
    "yogurt": 150
}

# Funzione per calcolare le calorie consumate
def calorie_consumate(cibo, quantita):
    if cibo not in cibo_calorie.keys():
        print("cibo non presente")
    elif cibo in cibo_calorie.keys():
        calorie_per_100g = cibo_calorie[cibo]
        calorie_totali = (calorie_per_100g / 100) * quantita
        return calorie_totali

# Funzione principale
def main():
    cibo_consumato = []

    while True:
        print("Menu:")
        print("\n 1. Aggiungi cibo consumato")
        print("\n 2. Calcola calorie totali")
        print("\n 3. Esci")

        scelta = input("Scegli un'opzione: ")

        if scelta == "1":
            print("\n")

            for key, value in cibo_calorie.items() :
                print (key, value)

            cibo = input("Inserisci il cibo consumato: ").lower()
            quantita = float(input("Inserisci la quantità (in grammi): "))
            cibo_consumato.append((cibo, quantita))

        elif scelta == "2":
            calorie_totali = sum(calorie_consumate(c, q) for c, q in cibo_consumato)
            print(f"\nCalorie totali consumate: {calorie_totali} calorie")
        elif scelta == "3":
            break
        else:
            print("\nScelta non valida. Riprova.")

    if __name__ == "__main__":
        main()
```

Menu:

1. Aggiungi cibo consumato
2. Calcola calorie totali
3. Esci

Scegli un'opzione: 1

```
pizza 285  
hamburger 250  
insalata 100  
pollo arrosto 335  
yogurt 150  
Inserisci il cibo consumato: pizza  
Inserisci la quantità (in grammi): 100
```

Menu:

1. Aggiungi cibo consumato
2. Calcola calorie totali
3. Esci

Scegli un'opzione: 2

```
Calorie totali consumate: 285.0 calorie  
Menu:
```

1. Aggiungi cibo consumato
2. Calcola calorie totali
3. Esci

Scegli un'opzione: 3

```
In [45]: acquisti={}
acquisti["pan bauletto"] = 10
acquisti["nutella"] = 10
```

```
In [46]: acquistidue={
    "pan bauletto": 10,
    "nutella": 10,
}
```

```
In [47]: acquistidue
```

```
Out[47]: {'pan bauletto': 10, 'nutella': 10}
```

```
In [48]: myTuple = ("John", "Peter", "Vicky")
x = ",".join(myTuple)

print(x,type(x))
```

```
John,Peter,Vicky <class 'str'>
```

```
In [ ]:
```

```
In [51]: my_tuple = ()
print(my_tuple)

my_tuple = (1, 2, 3)
print(my_tuple)

my_tuple = (1, "Hello", 3.4)
print(my_tuple)

my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

```
()  
(1, 2, 3)  
(1, 'Hello', 3.4)  
('mouse', [8, 4, 6], (1, 2, 3))
```

## Personaggi storie e romanzi

In [52]: `import random`

```
# Liste di specie, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilita = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione miniera"]

# Genera un personaggio casuale
specie = random.choice(speci)
classe = random.choice(classi)
arma = random.choice(armi)
abilita_scelte = random.sample(abilita, random.randint(1, 3))

# Stampa il personaggio generato
print(f"Personaggio Fantasy Generato:")
print(f"Specie: {specie}")
print(f"Classe: {classe}")
print(f"Arma: {arma}")
print(f"Abilità: {', '.join(abilita_scelte)}")
```

Personaggio Fantasy Generato:  
Specie: Orco  
Classe: Ladro  
Arma: Daga  
Abilità: Incantesimi di guarigione, Magia dell'acqua

In [53]: `import random`

```
# Liste di specie, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilita = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione mineraria"]

# Funzione per creare un personaggio casuale
def crea_personaggio():
    return {
        "Specie": random.choice(speci),
        "Classe": random.choice(classi),
        "Arma": random.choice(armi),
        "Abilità": random.sample(abilita, random.randint(1, 3))
    }

# Funzione principale
def main():
    personaggio_generato = crea_personaggio()
    print("Personaggio Fantasy Generato:")
    for chiave, valore in personaggio_generato.items():
        if chiave == "Abilità":
            valore = ', '.join(valore)
        print(f'{chiave}: {valore}')

# Eseguire la funzione "main" quando il programma viene eseguito
if __name__ == "__main__":
    main()
```

```
Personaggio Fantasy Generato:
Specie: Nano
Classe: Chierico
Arma: Daga
Abilità: Camuffamento, Estrazione mineraria
```

In [ ]:

In [54]:

```
# Liste di specie, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilita = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione mineraria"]

# Funzione per creare un personaggio casuale
def crea_personaggio():

    personaggio={
        "Specie": random.choice(speci),
        "Classe": random.choice(classi),
        "Arma": random.choice(armi),
        "Abilità": random.sample(abilita, random.randint(1, 3))
    }

    return personaggio

# Funzione principale
def main():
    personaggio_generato=crea_personaggio()

    print("Personaggio Fantasy Generato:")
    for chiave, valore in personaggio_generato.items():
        if chiave == "Abilità":
            valore = ', '.join(valore)
        print(f"{chiave}: {valore}")

# Eseguire la funzione "main" quando il programma viene eseguito
if __name__ == "__main__":
    main()
```

```
Personaggio Fantasy Generato:
Specie: Gnomo
Classe: Chierico
Arma: Spada
Abilità: Incantesimi di guarigione, Estrazione mineraria, Furtività
```

```
In [55]: import random

physical_traits = ["capelli neri", "capelli biondi", "occhi azzurri", "occhi marroni", "barba"]
personality_traits = ["gentile", "introverso", "estroverso", "ottimista", "pessimista"]
backgrounds = ["contadino", "nobile", "artigiano", "commerciale", "avventuriero"]
motivations = ["vendetta", "ricchezza", "amore", "vita eterna", "conoscenza"]

def genera_personaggio():
    nome = input("Inserisci il nome del personaggio: ")
    aspetto_fisico = random.choice(physical_traits)
    aspetto_personale = random.choice(personality_traits)
    sfondo = random.choice(backgrounds)
    motivazione = random.choice(motivations)

    descrizione = f"Nome: {nome}\nAspetto fisico: {aspetto_fisico}\nAspetto personale: {aspetto_personale}\nSfondo: {sfondo}\nMotivazione: {motivazione}"

    return descrizione

print("Generatore di Personaggi per Romanzi")
print(genera_personaggio())
```

```
Generatore di Personaggi per Romanzi
Inserisci il nome del personaggio: Pier
Nome: Pier
Aspetto fisico: sovrappeso
Aspetto personale: ambizioso
Sfondo: guerriero
Motivazione: fama
```

## LA LETTERATURA COMBINATORIA

```
In [56]: Anton
print("Generatore di Personaggi per Romanzi")
print(genera_personaggio())
```

```
Generatore di Personaggi per Romanzi
Inserisci il nome del personaggio: Tony
Nome: Tony
Aspetto fisico: labbra sottili
Aspetto personale: equilibrato
Sfondo: biologo
Motivazione: viaggiare nel tempo
```

```
In [ ]: # generatore di titoli nobiliari
```

```
In [ ]:
```

In [58]: `import random`

```
# Database di citazioni
citazioni = [
    "La vita è ciò che succede mentre sei occupato a fare altri progetti. -",
    "Il successo è camminare da un fallimento all'altro senza perdere l'entusiasmo.",
    "La felicità è quando ciò che pensi, ciò che dici e ciò che fai sono in sintonia tra loro.",
    "La vita è davvero semplice, ma insistiamo nel renderla complicata. - Confucio",
    "L'unico modo per fare un grande lavoro è amare quello che fai. - Steve Jobs",
    "La vita è 10% ciò che ci accade e 90% come reagiamo. - Charles R. Swindoll"
]

# Funzione per generare una citazione casuale
def genera_citazione():
    return random.choice(citazioni)

# Funzione principale
def main():
    print("Benvenuto nel Generatore di Citazioni!")
    input("Premi Invio per ottenere una citazione casuale...")

    citazione = genera_citazione()
    print(f"Citazione del giorno: {cita}zione")

if __name__ == "__main__":
    main()
```

```
Benvenuto nel Generatore di Citazioni!
Premi Invio per ottenere una citazione casuale...
Citazione del giorno: Il successo è camminare da un fallimento all'altro senza perdere l'entusiasmo. - Winston Churchill
```

## GENERATORE DI POST DA INFLUENCER

In [59]: `import random`

```
# Lista di frammenti di citazioni famose (più brevi)
frammenti = [
    "La vita è un'avventura.",
    "Il successo richiede impegno.",
    "Sii creativo.",
    "Non arrenderti mai.",
    "Semplicità ed eleganza.",
    "Ama ciò che fai.",
    "Fallo oggi.",
    "La saggezza del fallimento.",
    "Ogni giorno conta.",
    "Sii audace.",
    "Pensa diversamente.",
    "Credi in te stesso.",
    "La felicità è un viaggio.",
    "Sii il cambiamento che vuoi vedere.",
    "Non avere rimpianti.",
    "Sogna in grande.",
    "Abbraccia il caos.",
    "Lavora sodo, sogna in grande.",
    "Crescita personale.",
    "Sii gentile.",
    "L'arte di ascoltare.",
    "Inseguire i tuoi sogni.",
    "Non limitarti.",
    "Cambia il mondo.",
    "Fai la differenza.",
    "Il potere della positività.",
    "Trova la tua passione.",
    "Fai ciò che ami.",
    "Ogni giorno è un nuovo inizio.",
    "Rischiare è vivere.",
    "Perchè la conoscenza è potere.",
    "Tutto grazie al duro lavoro e alla fatica.",
    "Tutto grazie al duro lavoro e alla fatica.",
    "Questo è il segreto del successo!"
]

# Funzione per creare nuove citazioni rimescolando i frammenti
def crea_citazione():
    num_frammenti = random.randint(5, 7) # Scegli un numero casuale di frammenti
    citazione_rimescolata = random.sample(frammenti, num_frammenti)
    nuova_citazione = " ".join(citazione_rimescolata)
    return nuova_citazione

# Genera una nuova citazione
nuova_citazione = crea_citazione()
print("Nuova citazione generata:")
print(nuova_citazione)
```

Nuova citazione generata:

Pensa diversamente. Abbraccia il caos. Inseguire i tuoi sogni. Perchè la conoscenza è potere. Credi in te stesso. Tutto grazie al duro lavoro e alla fatica.



In [15]: `import random`

```
frammenti = [
    "La vita è un'avventura.",
    "Il successo richiede impegno.",
    "Sii creativo.",
    "Non arrenderti mai.",
    "Semplicità ed eleganza.",
    "Ama ciò che fai.",
    "Fallo oggi.",
    "La saggezza del fallimento.",
    "Ogni giorno conta.",
    "Sii audace.",
    "Pensa diversamente.",
    "Credi in te stesso.",
    "La felicità è un viaggio.",
    "Sii il cambiamento che vuoi vedere.",
    "Non avere rimpianti.",
    "Sogna in grande.",
    "Abbraccia il caos.",
    "Lavora sodo, sogna in grande.",
    "Crescita personale.",
    "Sii gentile.",
    "L'arte di ascoltare.",
    "Inseguire i tuoi sogni.",
    "Non limitarti.",
    "Cambia il mondo.",
    "Fai la differenza.",
    "Il potere della positività.",
    "Trova la tua passione.",
    "Fai ciò che ami.",
    "Ogni giorno è un nuovo inizio.",
    "Rischiare è vivere.",
    "Perchè la conoscenza è potere.",
    "Tutto grazie al duro lavoro e alla fatica.",
    "Tutto grazie al duro lavoro e alla fatica.",
    "Questo è il segreto del successo!",
    "La vita è una tela: dipingi il tuo capolavoro.",
    "Il futuro appartiene a coloro che credono nella bellezza dei propri sogni.",
    "Sii il cambiamento che desideri vedere nel mondo.",
    "La vita è troppo importante per essere presa sul serio.",
    "La perseveranza è la chiave del successo.",
    "Le opportunità non capitano, le crei tu.",
    "Non smettere mai di imparare.",
    "La gentilezza è la lingua che il sordo può sentire e il cieco può vedere.",
    "La creatività è l'intelligenza che si diverte.",
    "La tua unica limitazione è la tua immaginazione.",
    "L'unico modo per fare un grande lavoro è amare ciò che fai.",
    "La gratitudine è il segreto della felicità.",
    "La fiducia in se stessi è la chiave del successo.",
    "Il successo è camminare da un fallimento all'altro senza perdere l'entusiasmo.",
    "Ogni sogno inizia con una semplice decisione di provare.",
    "Il tempo è troppo lento per coloro che aspettano, troppo veloce per coloro che vivono nel momento presente.",
    "La vita è troppo breve per essere infelice.",
    "La vita è piena di sfide, ma ogni sfida porta con sé opportunità.",
    "La tua mentalità determina la tua realtà.",
    "Il cambiamento è la sola costante nella vita.",
    "Nessun giorno è uguale a un altro, ogni mattina porta con sé una benedizione.",
    "Se non riesci a farlo bene, almeno fallo con passione.",
    "Il successo non è definito da ciò che hai, ma da chi sei.",
    "La felicità è un'abitudine, coltivala."]
```

```

"Il segreto per ottenere ciò che vuoi è credere di meritarlo.",  

"Il coraggio è fare ciò che è giusto, non ciò che è facile.",  

"L'unico limite per il tuo futuro è la tua immaginazione.",  

"Sii la migliore versione di te stesso.",  

"Nessun sogno è troppo grande, nessun obiettivo è troppo lontano.",  

"La vita è una serie di momenti da godere.",  

"Vivi la tua vita senza rimpianti.",  

"La conoscenza è il tesoro più grande.",  

"La vita è un'opportunità, coglila.",  

"Sii la stella della tua vita.",  

"Non importa quanto sia difficile oggi, il domani sarà migliore.",  

"Ricorda sempre di sorridere.",  

"Le tue azioni parlano più forte delle tue parole.",  

"Non avere paura di fallire, abbi paura di non provare.",  

"La tua mente è un potente strumento, riempila di pensieri positivi.",  

"La gentilezza è una lingua che tutti possono capire.",  

"La saggezza viene dall'esperienza.",  

"Sii grato per ciò che hai e lavora duramente per ciò che desideri.",  

"Ogni giorno è una nuova opportunità per essere una persona migliore.",  

"Le tue azioni determinano il tuo destino.",  

"Il successo inizia con un solo passo.",  

"La vita è una preziosa avventura, sii pronto a esplorarla.",  

"Non aspettare il momento giusto, crea il momento giusto.",  

"Il futuro appartiene a coloro che credono nella bellezza dei propri sogni.",  

"La tua volontà è la chiave del tuo successo.",  

"Non smettere mai di sognare.",  

"Vivi la vita al massimo.",  

"L'amore è la forza più potente del mondo.",  

"La gratitudine è una medicina per l'anima.",  

"Il successo richiede sacrificio.",  

"La fiducia in se stessi è il primo segreto del successo.",  

"Il miglior modo per prevedere il futuro è crearlo.",  

"Non puoi cambiare il passato, ma puoi influenzare il futuro.",  

"Le persone più felici non hanno tutto, ma fanno il meglio di tutto ciò.",  

"Sii il tuo più grande sostenitore.",  

"Non importa quanto sia difficile, non arrenderti mai.",  

"La vita è fatta di piccoli momenti.",  

"La bellezza è ovunque, basta saperla vedere.",  

"Non c'è mai un momento perfetto per iniziare, inizia ora.",  

"Sii grato per ogni giorno che ti è stato regalato.",  

"La vita è piena di sorprese, abbracciale.",  

"Sii la migliore versione di te stesso ogni giorno.",  

"Il successo è il risultato di una mentalità positiva.",  

"La perseveranza è la chiave del successo.",  

"Il coraggio è la forza per affrontare le sfide.",  

"Il futuro appartiene a coloro che credono nella bellezza dei propri sogni.",  

"Il cambiamento è l'inizio di una nuova avventura.",  

"La tua mentalità determina la tua realtà.",  

"Sii il cambiamento che vuoi vedere nel mondo.",  

"L'unico modo per ottenere ciò che vuoi è credere di meritarlo."
]
```

```

# Funzione per creare nuove citazioni rimescolando i frammenti
def crea_citazione():
    num_frammenti = random.randint(4, 7) # Scegli un numero casuale di frammenti
    citazione_rimescolata = random.sample(frammenti, num_frammenti)
    nuova_citazione = " ".join(citazione_rimescolata)
    return nuova_citazione

# Genera una nuova citazione

```

```
def main():
    nuova_citazione = crea_citazione()
    print("Nuova citazione generata:")
    print(nuova_citazione)

if __name__ == "__main__":
    main()
```

Nuova citazione generata:

Non puoi cambiare il passato, ma puoi influenzare il futuro. Ricorda sempre di sorridere. L'arte di ascoltare. La vita è un'avventura. L'unico modo per ottenere ciò che vuoi è credere di meritarlo. Il successo è il risultato di una mentalità positiva. Sii audace.

```
In [60]: import random
#Liste di parole predefinite per la generazione della poesia
aggettivi = ["dolce", "sereno", "profondo", "luminoso", "gentile"]
sostantivi = ["amore", "mare", "cielo", "vento", "sogno"]
verbi = ["danza", "splende", "abbraccia", "canta", "sorride"]

#Genera una poesia casuale
def genera_poesia():
    verso1 = f"Il {random.choice(aggettivi)} {random.choice(sostantivi)} {random.choice(verbi)}."
    verso2 = f"Il {random.choice(aggettivi)} {random.choice(sostantivi)} {random.choice(verbi)}."
    verso3 = f"Nel {random.choice(sostantivi)} {random.choice(verbi)} con {random.choice(aggettivi)} {random.choice(sostantivi)}.""

    return f"{verso1}\n{verso2}\n{verso3}"
#Stampa la poesia generata
print(genera_poesia())
```

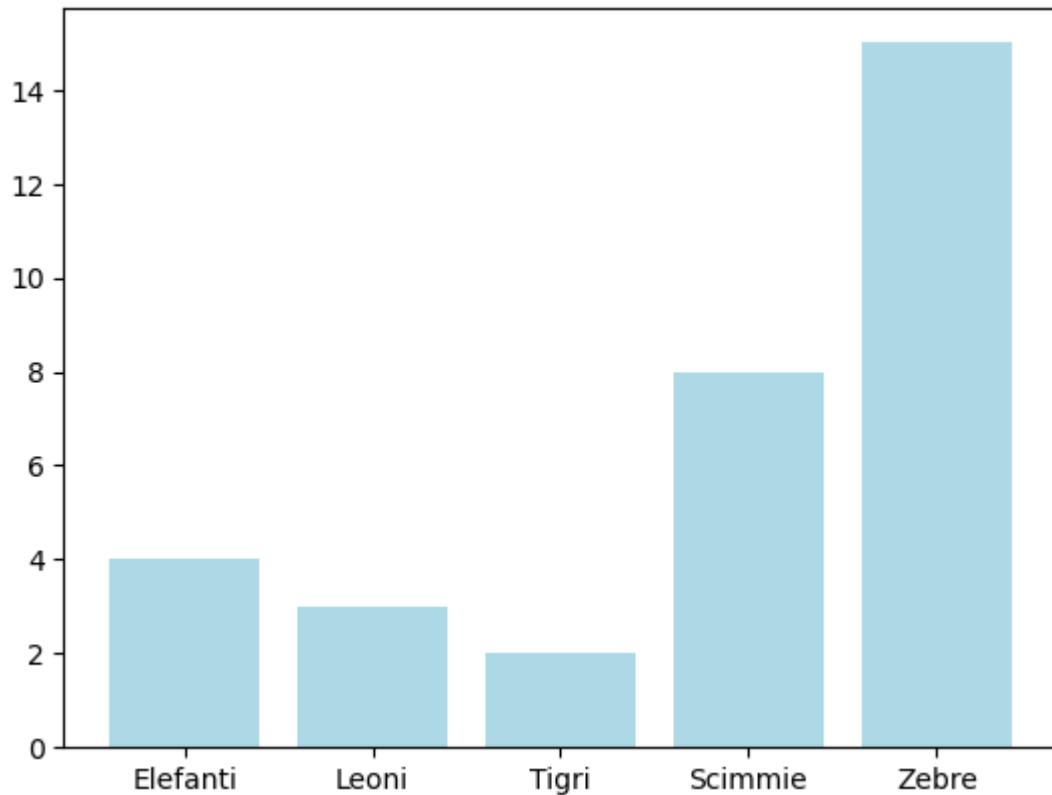
Il dolce vento danza.  
Il sereno sogno danza.  
Nel cielo sorride con sereno amore.

In [ ]:

In [ ]:

In [ ]:

```
In [2]: import matplotlib.pyplot as plt  
animali = ['Elefanti', 'Leoni', 'Tigri', 'Scimmie', 'Zebre']  
numero_animali = [4, 3, 2, 8, 15]  
plt.bar(animali, numero_animali, color="lightblue")  
plt.show()
```

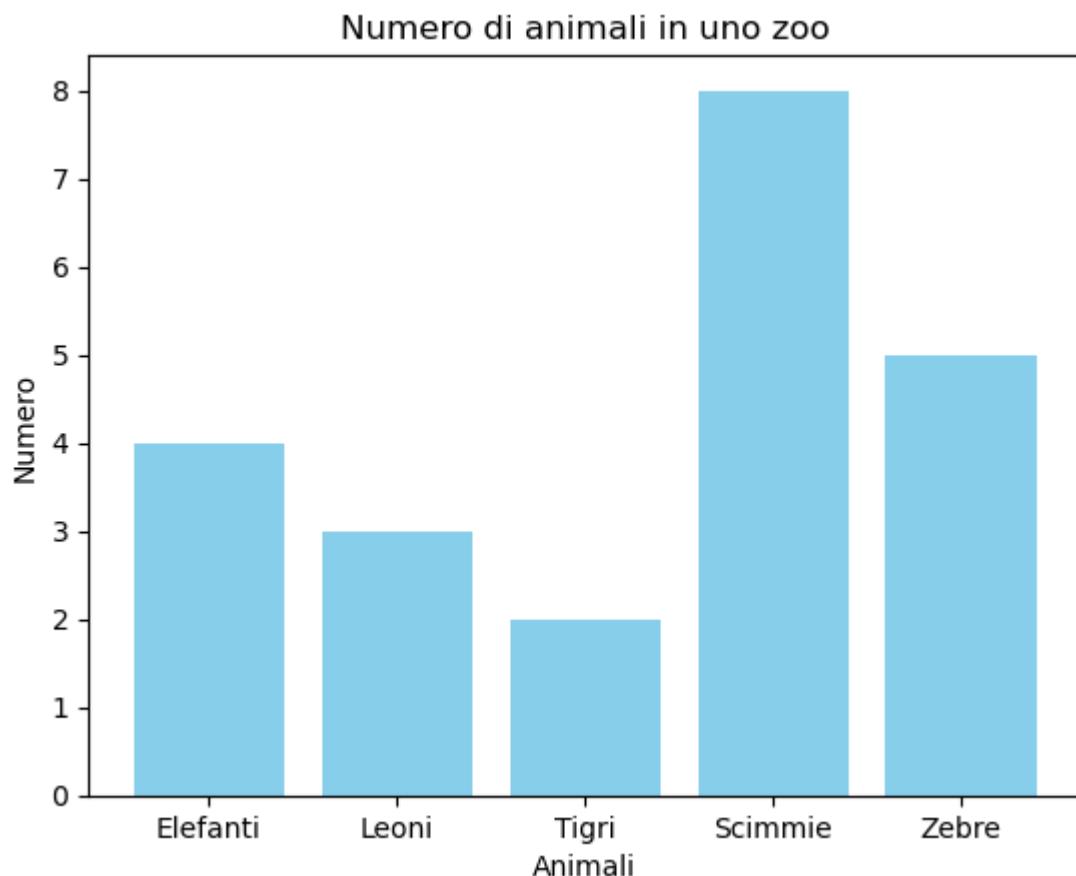


```
In [ ]:
```

```
In [2]: import matplotlib.pyplot as plt

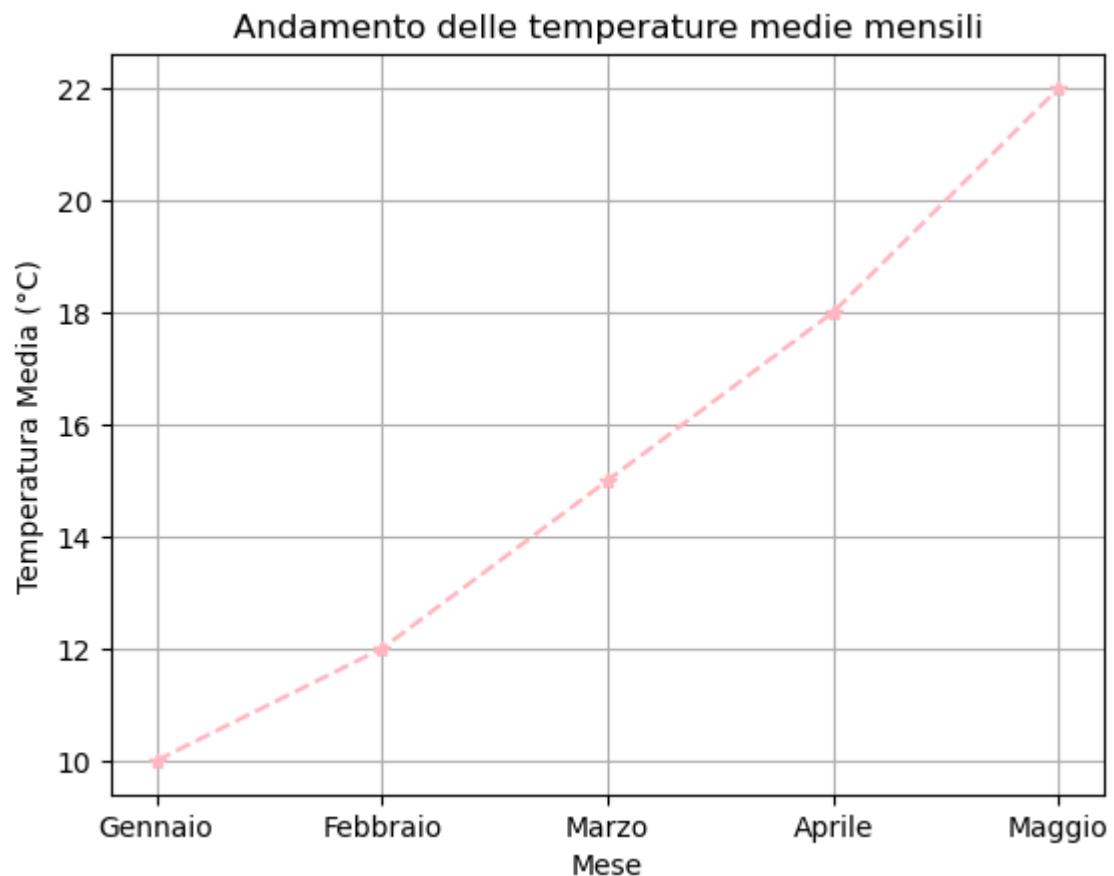
animali = ['Elefanti', 'Leoni', 'Tigri', 'Scimmie', 'Zebre']
numero_animali = [4, 3, 2, 8, 5]

plt.bar(animali, numero_animali, color='skyblue')
plt.title('Numero di animali in uno zoo')
plt.xlabel('Animali')
plt.ylabel('Numero')
plt.show()
```

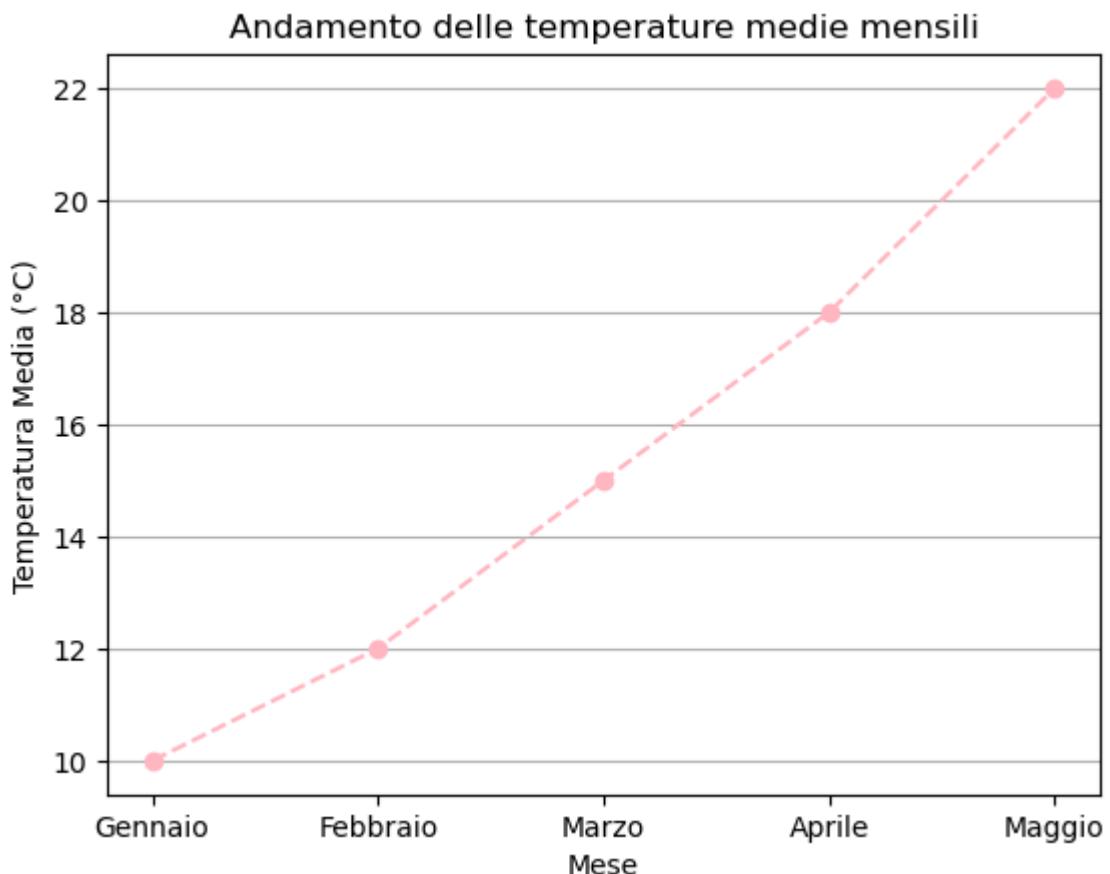


```
In [4]: mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]
plt.plot(mese, temperatura_media, marker='*', linestyle='--', color='lightpink')
plt.title('Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')
plt.grid(True)

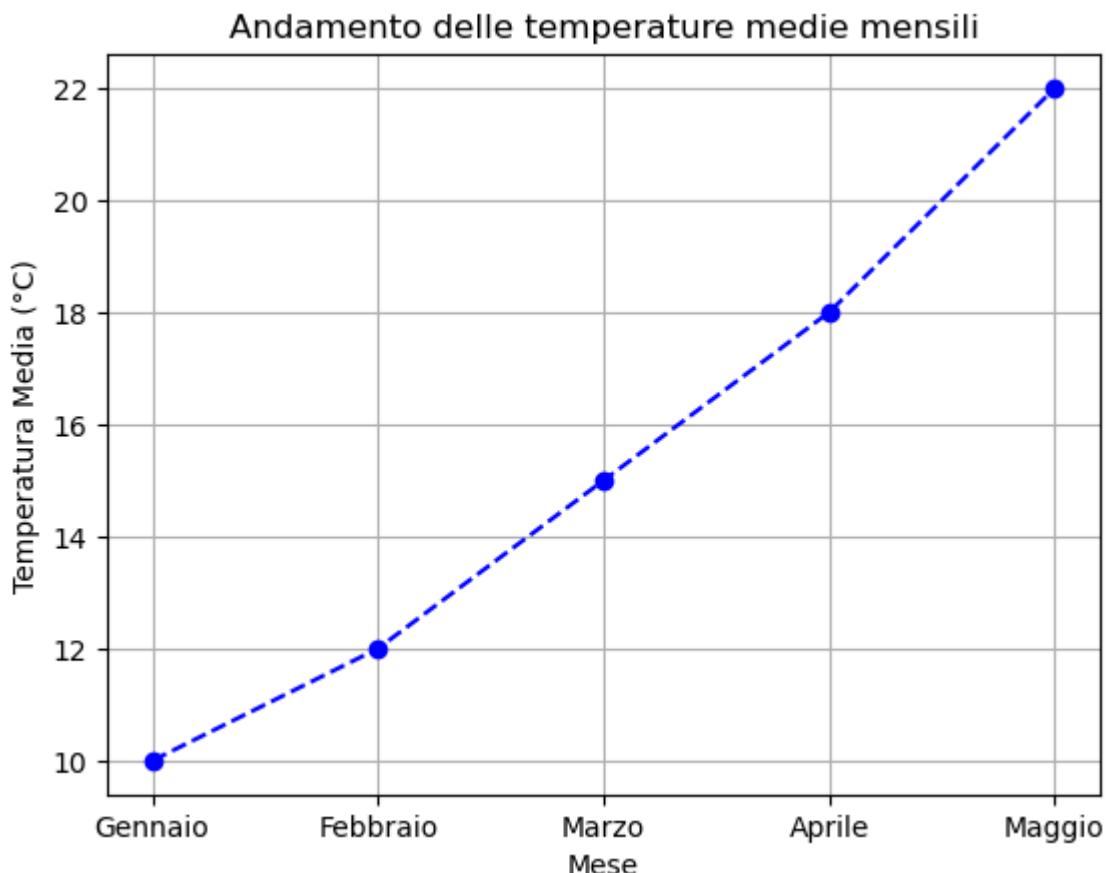
plt.show()
```



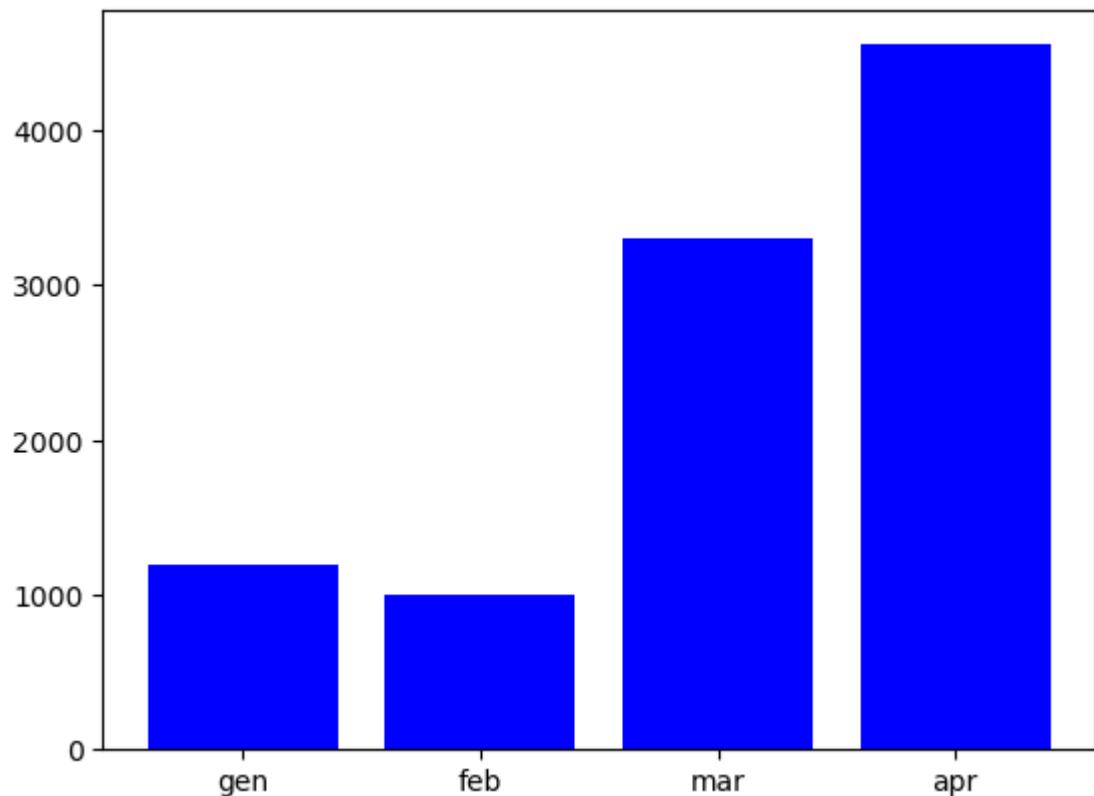
```
In [4]: mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]
plt.plot(mese, temperatura_media, marker='o', linestyle='--', color='lightpink')
plt.title('Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')
plt.grid(True, axis="y")
plt.show()
```



```
In [11]: mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]
plt.plot(mese, temperatura_media, marker='o', linestyle='--', color='blue')
plt.title('Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')
plt.grid(True)
plt.show()
```

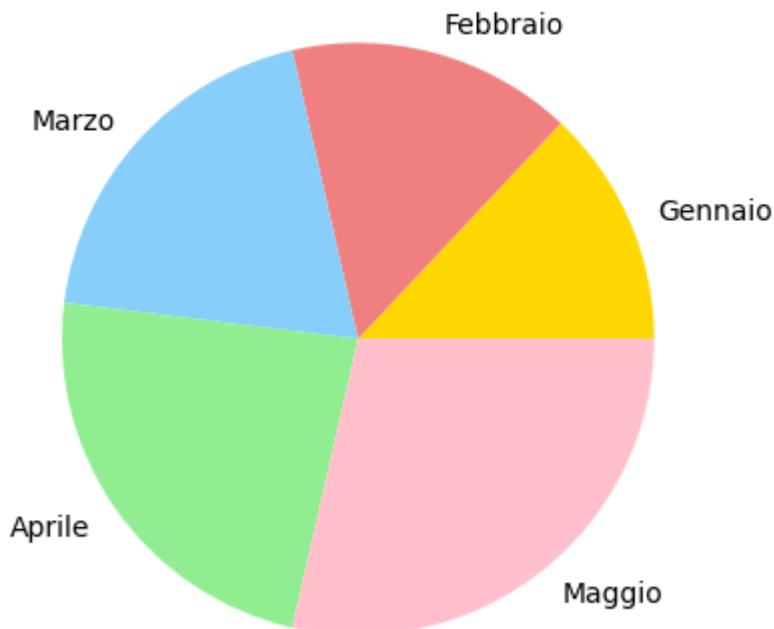


```
In [8]: vendite_mensili={  
    "gen":1200,  
    "feb":1000,  
    "mar":3300,  
    "apr":4555  
}  
  
plt.bar(vendite_mensili.keys(),vendite_mensili.values(),color="blue")  
plt.show()
```



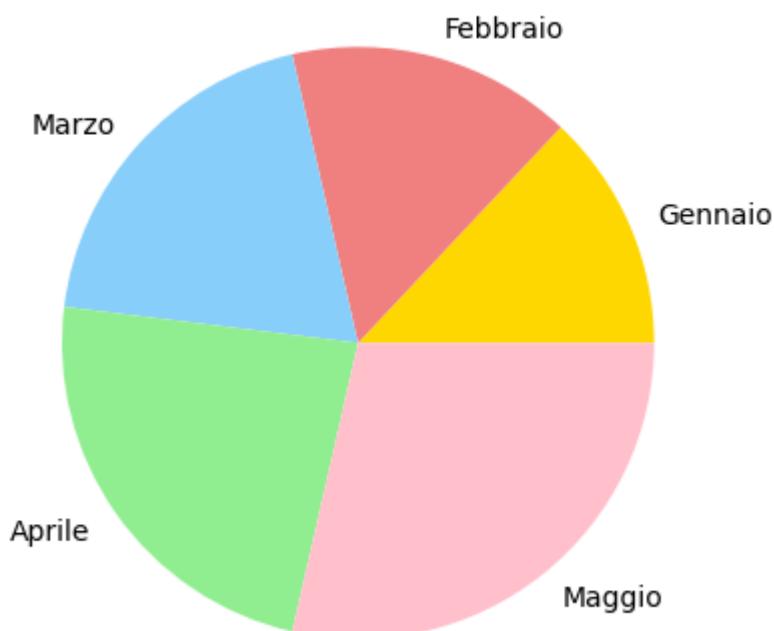
```
In [15]: colori = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']
mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]
plt.pie(temperatura_media, labels=mese, colors=colori)
plt.title('Percentuale di temperatura media mensile')
plt.show()
```

Percentuale di temperatura media mensile



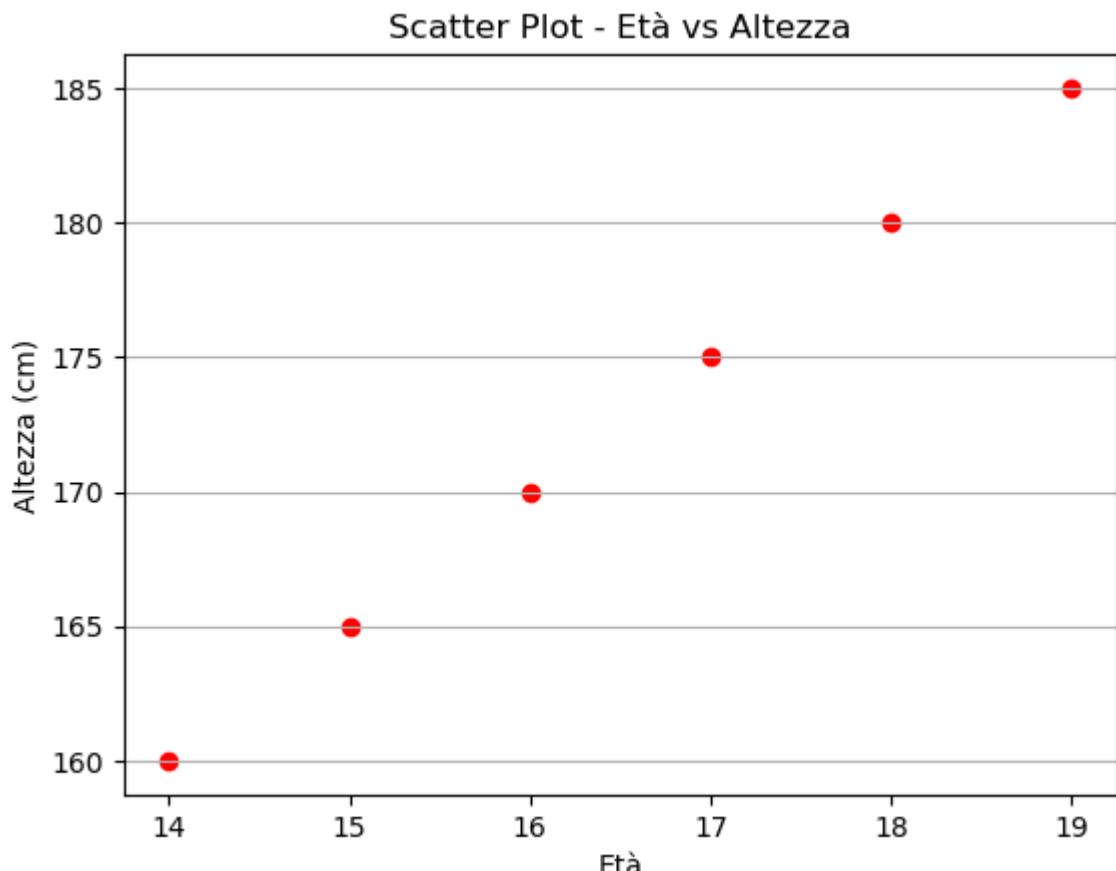
```
In [10]: colori = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']
temperatura_mesi={
    'Gennaio':10,
    'Febbraio':12,
    'Marzo':15,
    'Aprile':18,
    'Maggio':22
}
plt.pie(temperatura_mesi.values(), labels=temperatura_mesi.keys(), colors=colori)
plt.title('Percentuale di temperatura media mensile')
plt.show()
```

Percentuale di temperatura media mensile



```
In [ ]:
```

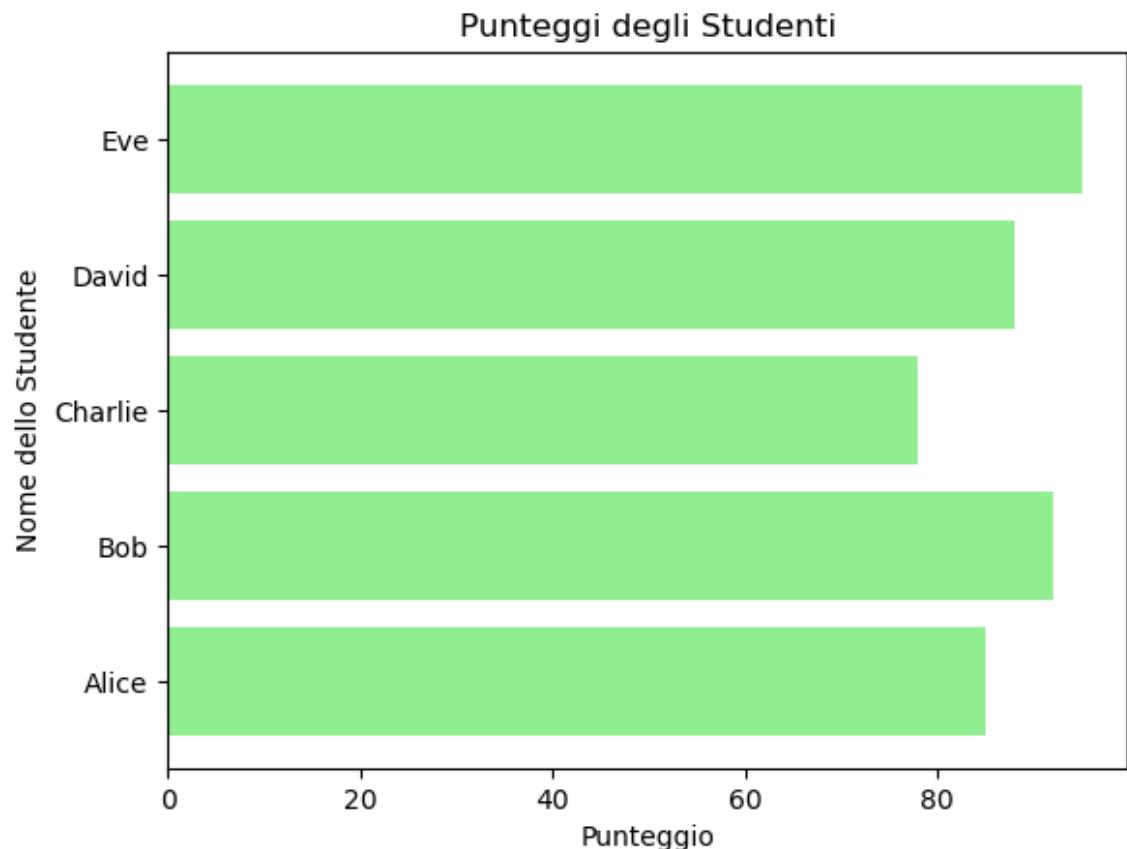
```
In [11]: età = [14, 15, 16, 17, 18, 19]
altezza = [160, 165, 170, 175, 180, 185]
plt.scatter(età, altezza, color='red', marker='o')
plt.title('Scatter Plot - Età vs Altezza')
plt.xlabel('Età')
plt.ylabel('Altezza (cm)')
plt.grid(True, axis='y')
plt.show()
```



```
In [ ]:
```

```
In [12]: nomi_studenti = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
punteggi = [85, 92, 78, 88, 95]

plt.barh(nomi_studenti, punteggi, color='lightgreen')
plt.title('Punteggi degli Studenti')
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')
plt.show()
```



```
In [13]: import matplotlib.pyplot as plt
import pandas as pd

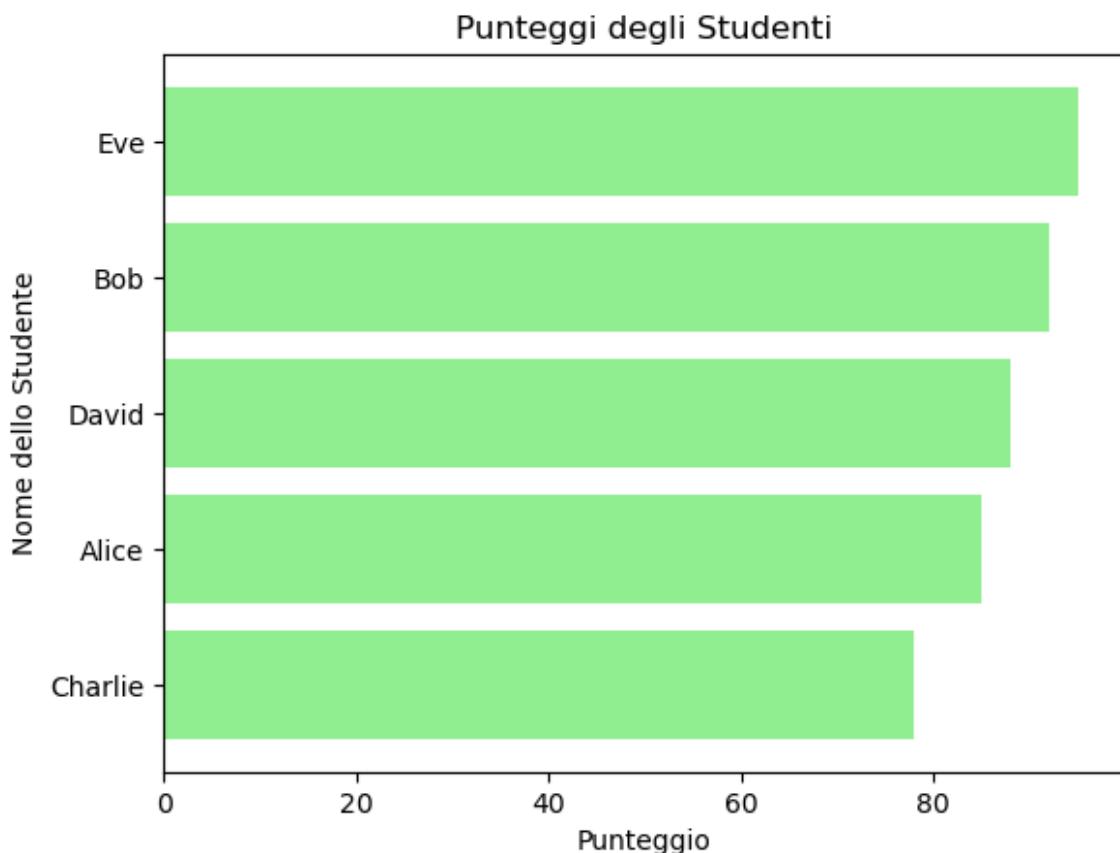
nomi_studenti = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
punteggi = [85, 92, 78, 88, 95]

# Crea un DataFrame con nomi e punteggi
data = {'Nome dello Studente': nomi_studenti, 'Punteggio': punteggi}
df = pd.DataFrame(data)
# Ordina il DataFrame per punteggio in ordine crescente
df.sort_values(by='Punteggio', inplace=True)
df
```

Out[13]:

	Nome dello Studente	Punteggio
2	Charlie	78
0	Alice	85
3	David	88
1	Bob	92
4	Eve	95

```
In [19]: plt.barh(df['Nome dello Studente'], df['Punteggio'], color='lightgreen')
plt.title('Punteggi degli Studenti')
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')
plt.show()
```



In [ ]:

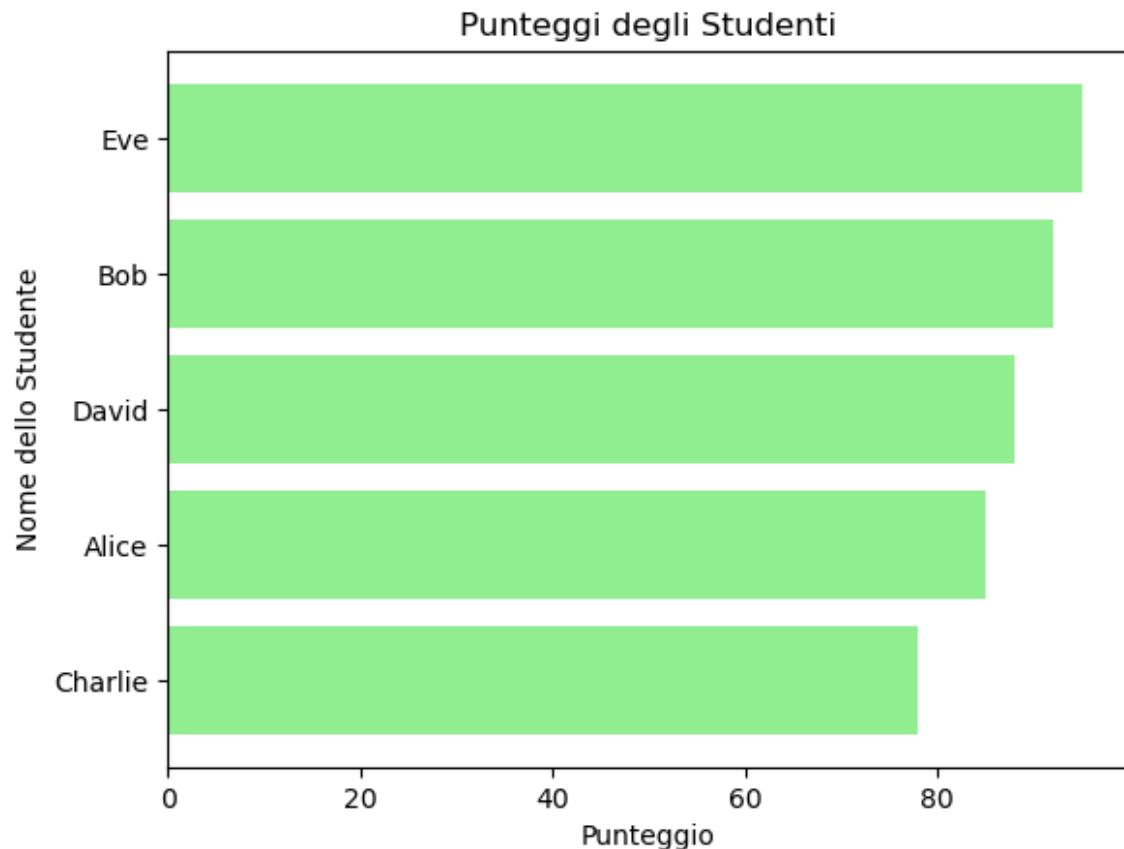
```
In [12]: import matplotlib.pyplot as plt
import pandas as pd

nomi_studenti = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
punteggi = [85, 92, 78, 88, 95]

# Crea un DataFrame con nomi e punteggi
data = {'Nome dello Studente': nomi_studenti, 'Punteggio': punteggi}
df = pd.DataFrame(data)

# Ordina il DataFrame per punteggio in ordine crescente
df1=df.sort_values(by='Punteggio', inplace=False)

plt.barh(df1['Nome dello Studente'], df1['Punteggio'], color='lightgreen')
plt.title('Punteggi degli Studenti')
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')
plt.show()
```



In [ ]:

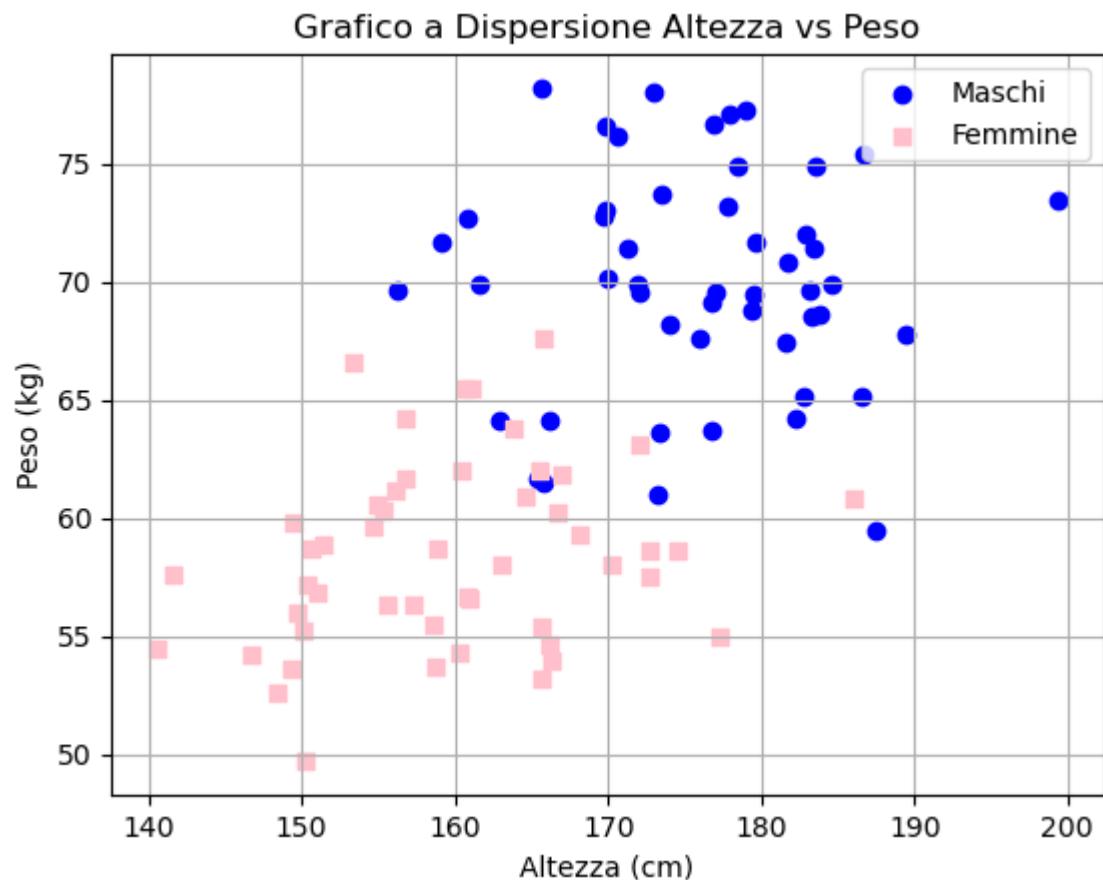
```
In [17]: import matplotlib.pyplot as plt
import numpy as np
# Dati di esempio
altezza_maschi = np.random.normal(175, 10, 50) # Altezza dei maschi
peso_maschi = np.random.normal(70, 5, 50) # Peso dei maschi

altezza_femmine = np.random.normal(162, 8, 50) # Altezza delle femmine
peso_femmine = np.random.normal(58, 4, 50) # Peso delle femmine
# Crea il grafico a dispersione per i maschi
plt.scatter(altezza_maschi, peso_maschi, color='blue', label='Maschi', marker='o')

# Crea il grafico a dispersione per le femmine
plt.scatter(altezza_femmine, peso_femmine, color='pink', label='Femmine', marker='s')

# Personalizza il grafico
plt.title('Grafico a Dispersione Altezza vs Peso')
plt.xlabel('Altezza (cm)')
plt.ylabel('Peso (kg)')
plt.legend(loc='upper right')
plt.grid(True)

# Mostra il grafico
plt.show()
```



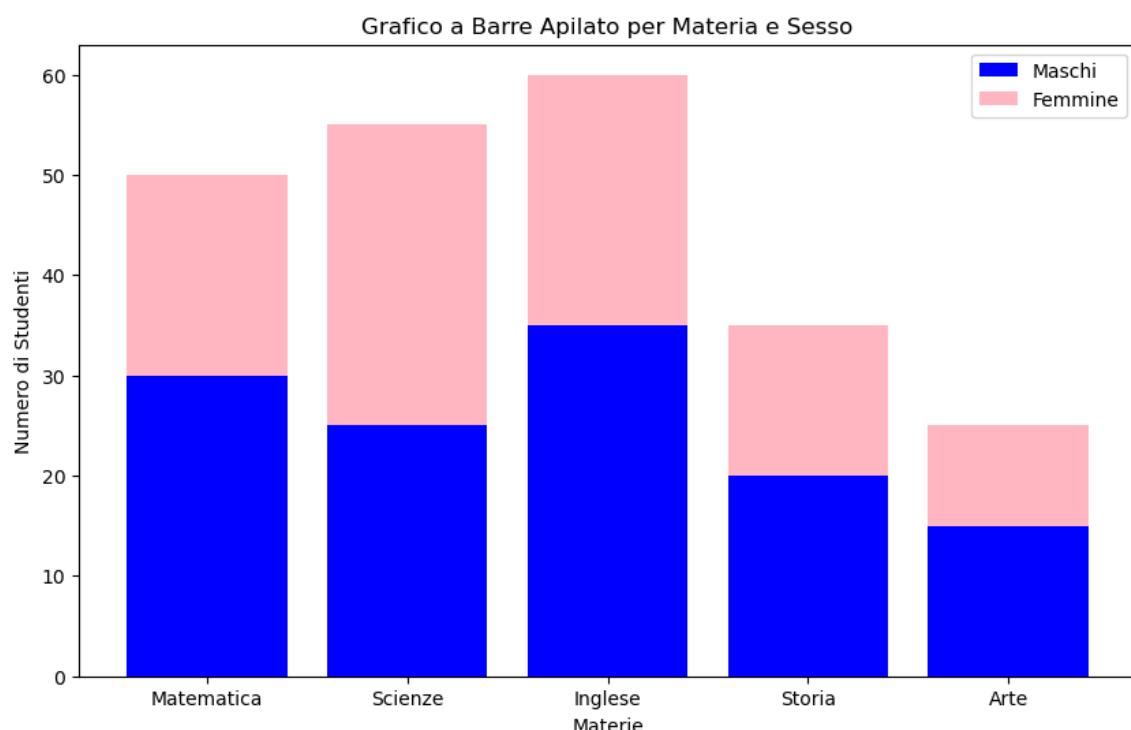
In [ ]:

In [ ]:

```
In [5]: # Dati di esempio
materie = ['Matematica', 'Scienze', 'Inglese', 'Storia', 'Arte']
maschi = [30, 25, 35, 20, 15] # Numero di studenti maschi per materia
femmine = [20, 30, 25, 15, 10] # Numero di studentesse per materia
# Creare il grafico a barre apilato
plt.figure(figsize=(10, 6)) # Imposta le dimensioni del grafico
plt.bar(materie, maschi, label='Maschi', color='blue')
plt.bar(materie, femmine, label='Femmine', bottom=maschi, color='lightpink')

# Personalizzare il grafico
plt.title('Grafico a Barre Apilato per Materia e Sesso')
plt.xlabel('Materie')
plt.ylabel('Numero di Studenti')
plt.legend(loc='upper right')

# Mostra il grafico
plt.show()
```



In [ ]:

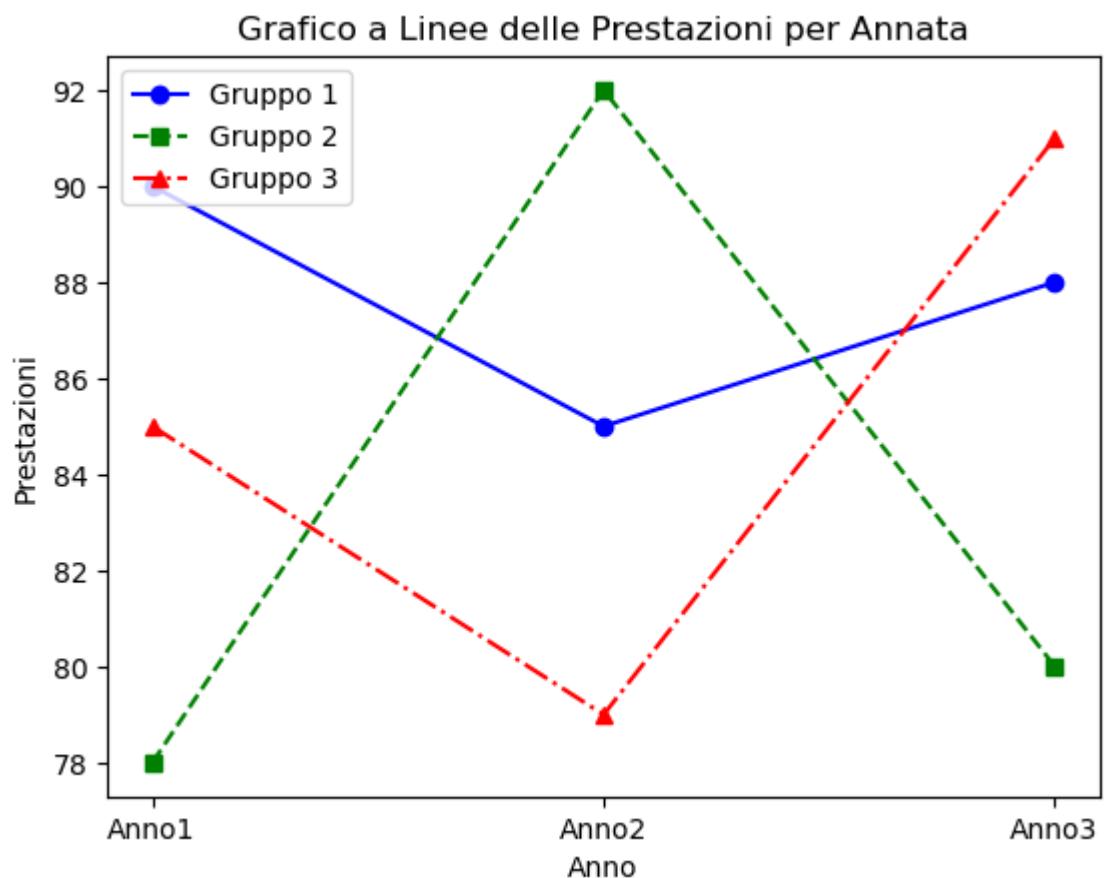
In [ ]:

In [ ]:

```
In [19]: annata = [ 'Anno1', 'Anno2', 'Anno3' ]
gruppo1 = [90, 85, 88]
gruppo2 = [78, 92, 80]
gruppo3 = [85, 79, 91]
plt.plot(annata, gruppo1, marker='o', label='Gruppo 1', linestyle='-', color='blue')
plt.plot(annata, gruppo2, marker='s', label='Gruppo 2', linestyle='--', color='green')
plt.plot(annata, gruppo3, marker='^', label='Gruppo 3', linestyle='-.', color='red')

plt.title('Grafico a Linee delle Prestazioni per Annata')
plt.xlabel('Anno')
plt.ylabel('Prestazioni')
plt.legend(loc='upper left')

plt.show()
```



```
In [ ]:
```

```
In [3]: import matplotlib.pyplot as plt
import numpy as np

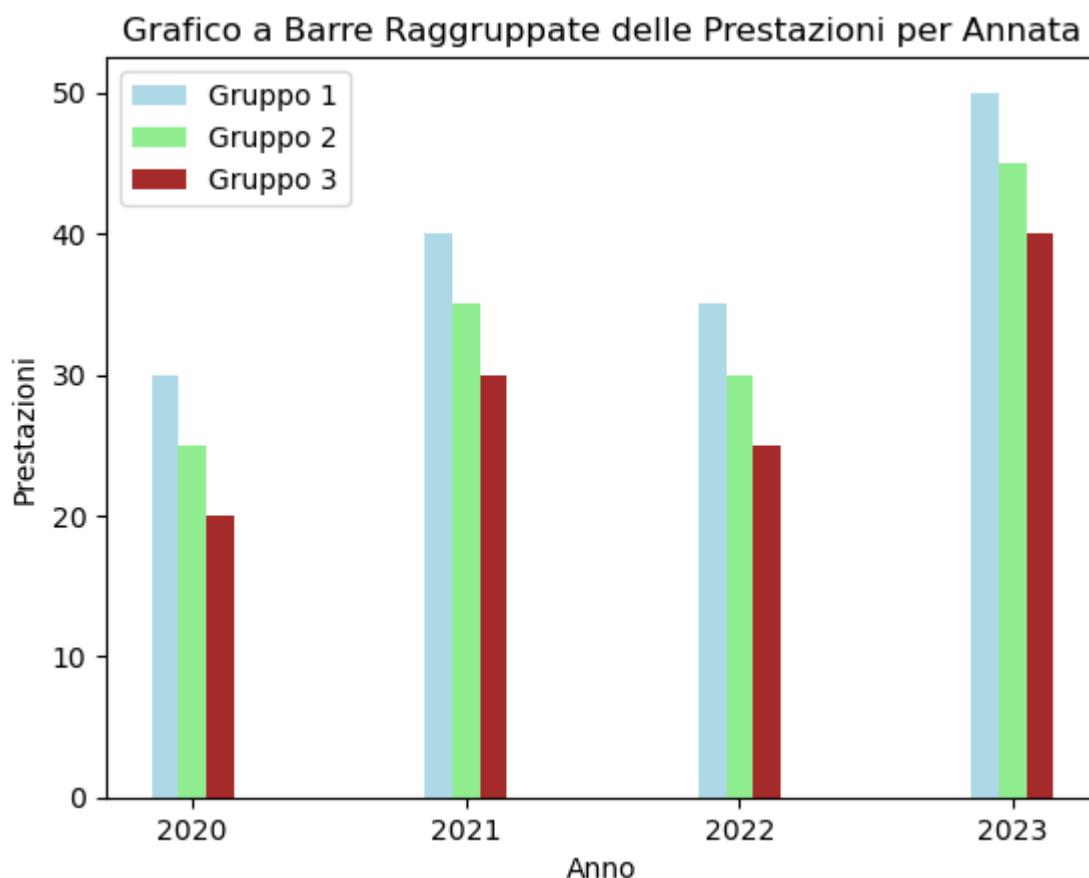
# Dati di esempio
annata = ['2020', '2021', '2022', '2023']
gruppo1 = [30, 40, 35, 50]
gruppo2 = [25, 35, 30, 45]
gruppo3 = [20, 30, 25, 40]

larghezza_barre = 0.1
indici = np.arange(len(annata))

plt.bar(indici - larghezza_barre, gruppo1, width=larghezza_barre, label='Gruppo 1')
plt.bar(indici, gruppo2, width=larghezza_barre, label='Gruppo 2', color='lightgreen')
plt.bar(indici + larghezza_barre, gruppo3, width=larghezza_barre, label='Gruppo 3')

plt.title('Grafico a Barre Raggruppate delle Prestazioni per Annata')
plt.xlabel('Anno')
plt.ylabel('Prestazioni')
plt.xticks(indici, annata)
plt.legend(loc='upper left')

plt.show()
```



```
In [4]: indici = np.arange(len(annata))
```

```
In [6]: indici - larghezza_barre
```

```
Out[6]: array([-0.1,  0.9,  1.9,  2.9])
```

```
In [26]: import matplotlib.pyplot as plt
import numpy as np
```

```
# Passo 2: Crea dati di esempio
```

```
mesi = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
```

```
vendite_prodotto_A = [100, 120, 90, 110, 95]
```

```
vendite_prodotto_B = [80, 95, 110, 85, 120]
```

```
vendite_prodotto_C = [60, 75, 70, 80, 65]
```

```
# Passo 3: Crea un grafico a barre empilato
```

```
plt.bar(mesi, vendite_prodotto_A, label='Prodotto A', color='blue')
```

```
plt.bar(mesi, vendite_prodotto_B, label='Prodotto B', color='green', bottom=vendite_prodotto_A)
```

```
plt.bar(mesi, vendite_prodotto_C, label='Prodotto C', color='red', bottom=vendite_prodotto_B)
```

```
# Passo 4: Personalizza il grafico
```

```
plt.title('Vendite Mensili per Prodotto')
```

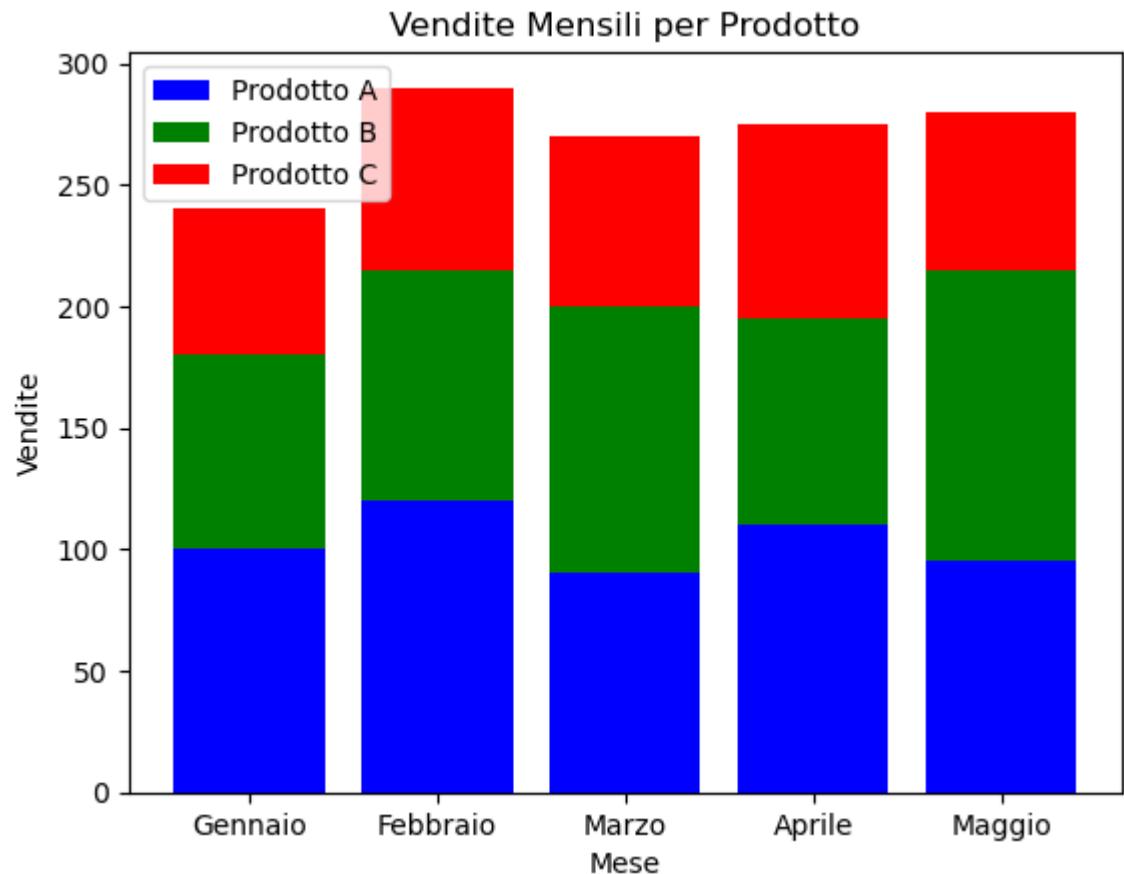
```
plt.xlabel('Mese')
```

```
plt.ylabel('Vendite')
```

```
plt.legend(loc='upper left')
```

```
# Passo 5: Mostra il grafico risultante
```

```
plt.show()
```



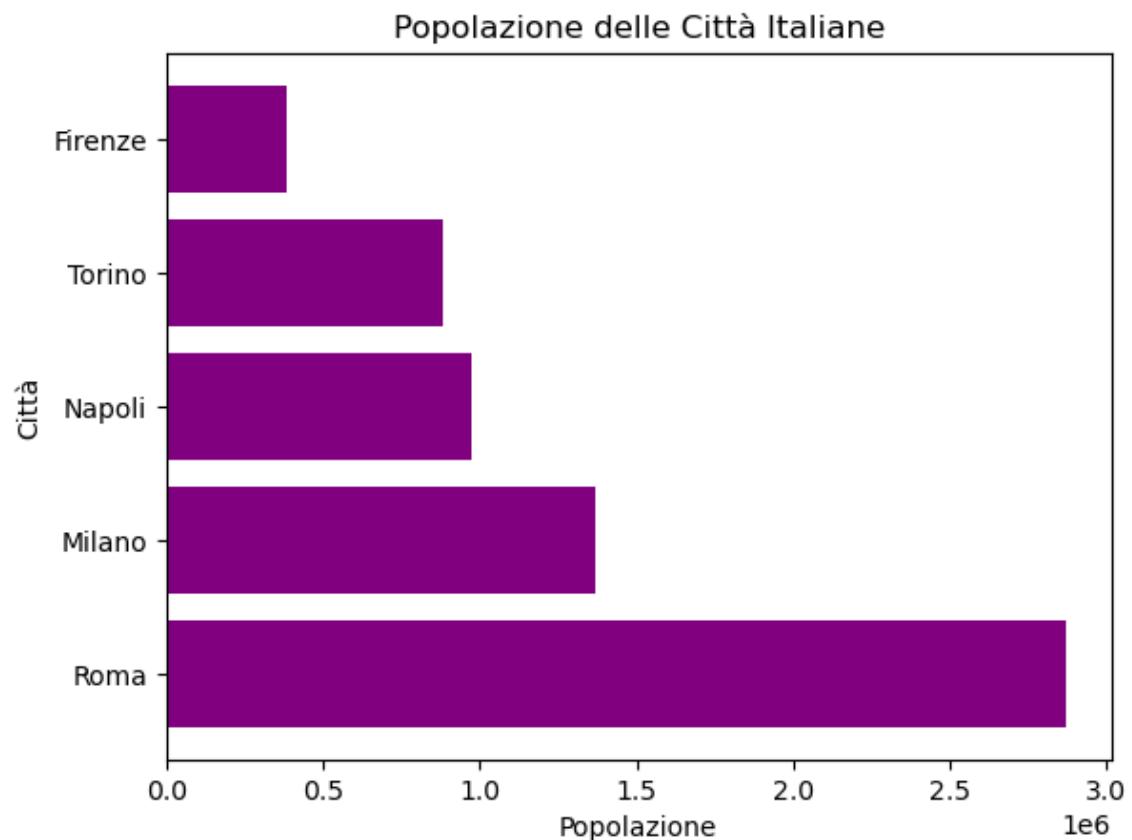
In [ ]:

```
In [112]: import matplotlib.pyplot as plt
import numpy as np

# Passo 2: Crea dati di esempio
città = ['Roma', 'Milano', 'Napoli', 'Torino', 'Firenze']
popolazione = [2870433, 1366180, 972198, 883767, 382258]

# Passo 3: Crea un grafico a barre orizzontali
plt.barh(città, popolazione, color='purple')
plt.title('Popolazione delle Città Italiane')
plt.xlabel('Popolazione')
plt.ylabel('Città')

# Passo 4: Mostra il grafico risultante
plt.show()
```



In [ ]:

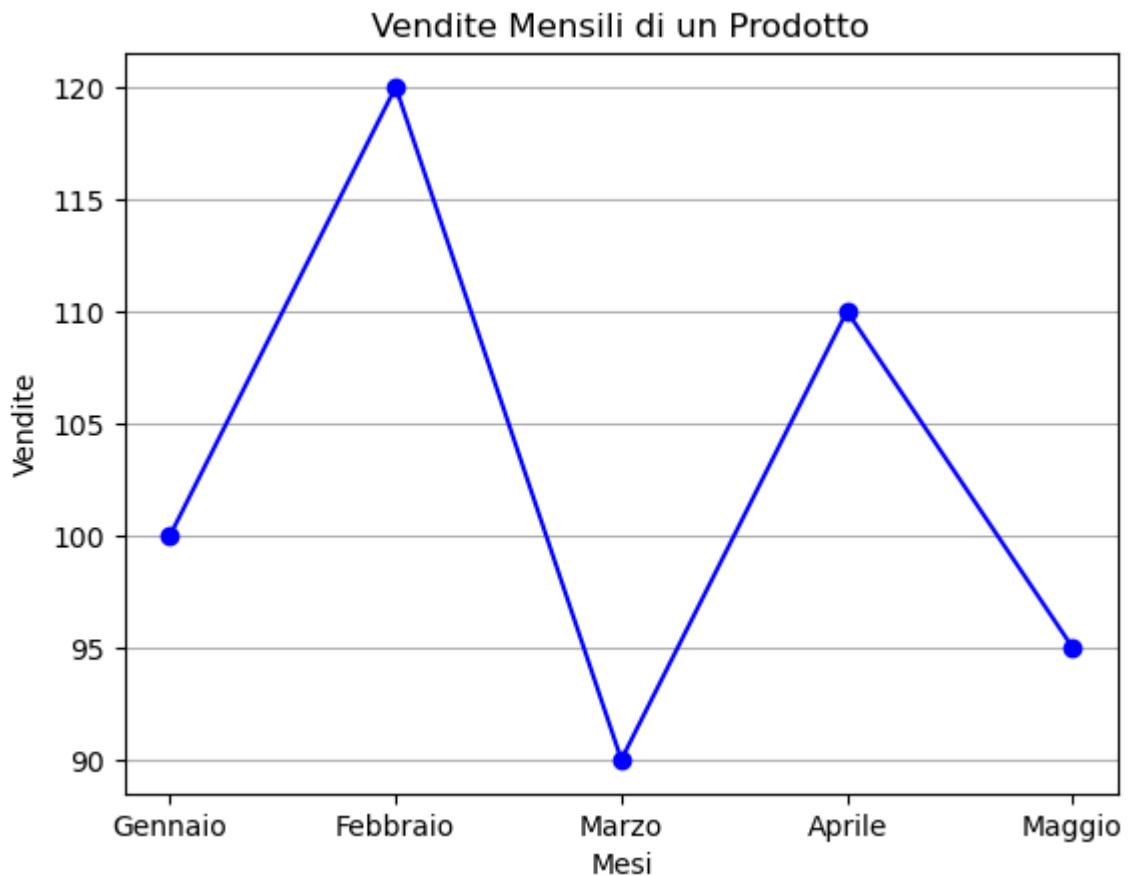
```
In [5]: import matplotlib.pyplot as plt
import numpy as np

# Passo 2: Crea dati di esempio
mesi = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
vendite = [100, 120, 90, 110, 95]

# Passo 3: Crea un grafico a linee
plt.plot(mesi, vendite, marker='o', linestyle='--', color='blue')

# Passo 4: Personalizza il grafico
plt.title('Vendite Mensili di un Prodotto')
plt.xlabel('Mesi')
plt.ylabel('Vendite')
plt.grid(True, axis="y")

# Passo 5: Mostra il grafico risultante
plt.show()
```



In [ ]:

In [ ]:

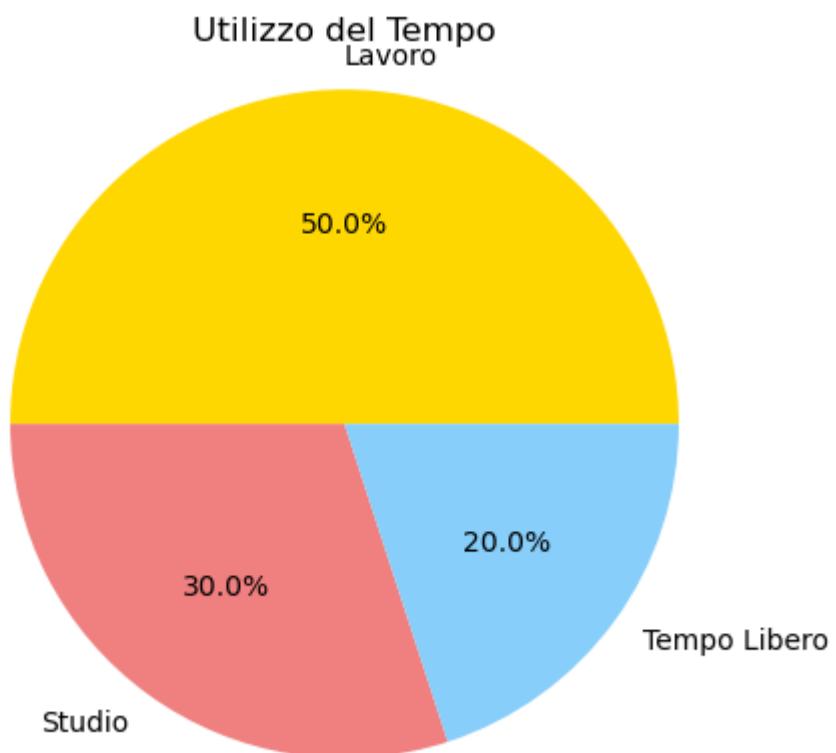
```
In [11]: import matplotlib.pyplot as plt

# Passo 2: Crea dati di esempio
attività = ['Lavoro', 'Studio', 'Tempo Libero']
percentuali = [50, 30, 20]
colori = ['gold', 'lightcoral', 'lightskyblue']

# Passo 3: Crea un grafico a torta
plt.pie(percentuali, labels=attività, colors=colori, autopct='%1.1f%%')

# Passo 4: Personalizza il grafico
plt.title('Utilizzo del Tempo')
plt.axis('equal') # Rendi il grafico a torta circolare

# Passo 5: Mostra il grafico risultante
plt.show()
```



```
In [ ]:
```

```
In [35]: import matplotlib.pyplot as plt
import numpy as np

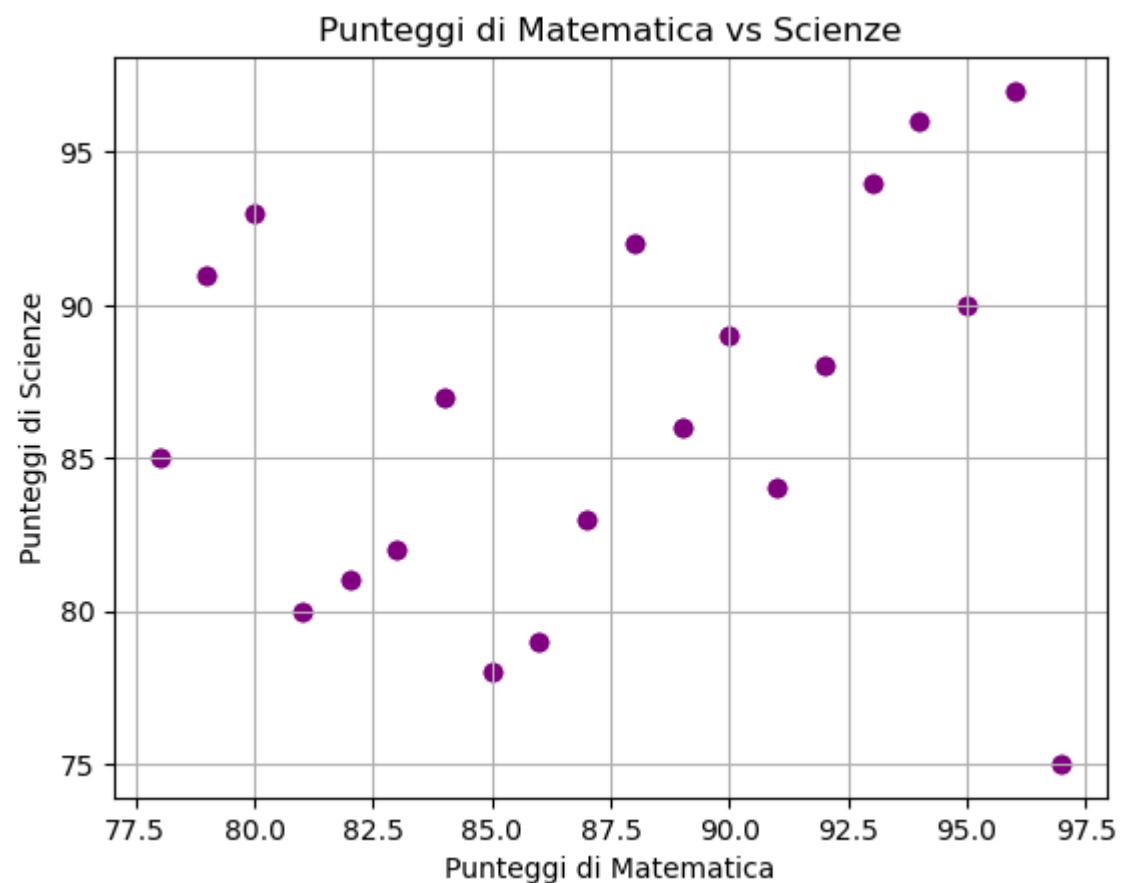
# Passo 2: Crea dati di esempio
punteggi_matematica = [85, 92, 78, 88, 95, 90, 89, 86, 79, 91, 84, 87, 83, 81, 96, 75, 80, 82, 87, 85, 93, 97, 90, 88, 86, 89, 92, 84, 87, 83, 82, 81]

punteggi_scienze = [78, 88, 85, 92, 90, 89, 86, 79, 91, 84, 87, 83, 82, 81, 97, 75, 80, 82, 87, 85, 93, 96, 90, 88, 86, 89, 92, 84, 87, 83, 82, 81]

# Passo 3: Crea un grafico a dispersione
plt.scatter(punteggi_matematica, punteggi_scienze, color='purple', marker='o')

# Passo 4: Personalizza il grafico
plt.title('Punteggi di Matematica vs Scienze')
plt.xlabel('Punteggi di Matematica')
plt.ylabel('Punteggi di Scienze')
plt.grid(True)

# Passo 5: Mostra il grafico risultante
plt.show()
```



In [36]: `import random`

```
punteggi_matematica = []
# Set a length of the list to 10
for i in range(0, 50):
    punteggi_matematica.append(random.randint(70, 100))
```

```
punteggi_scienze = []
# Set a length of the list to 10
for i in range(0, 50):
    # any random numbers from 0 to 1000
    punteggi_scienze.append(random.randint(70, 100))
```

# Passo 3: Crea un grafico a dispersione

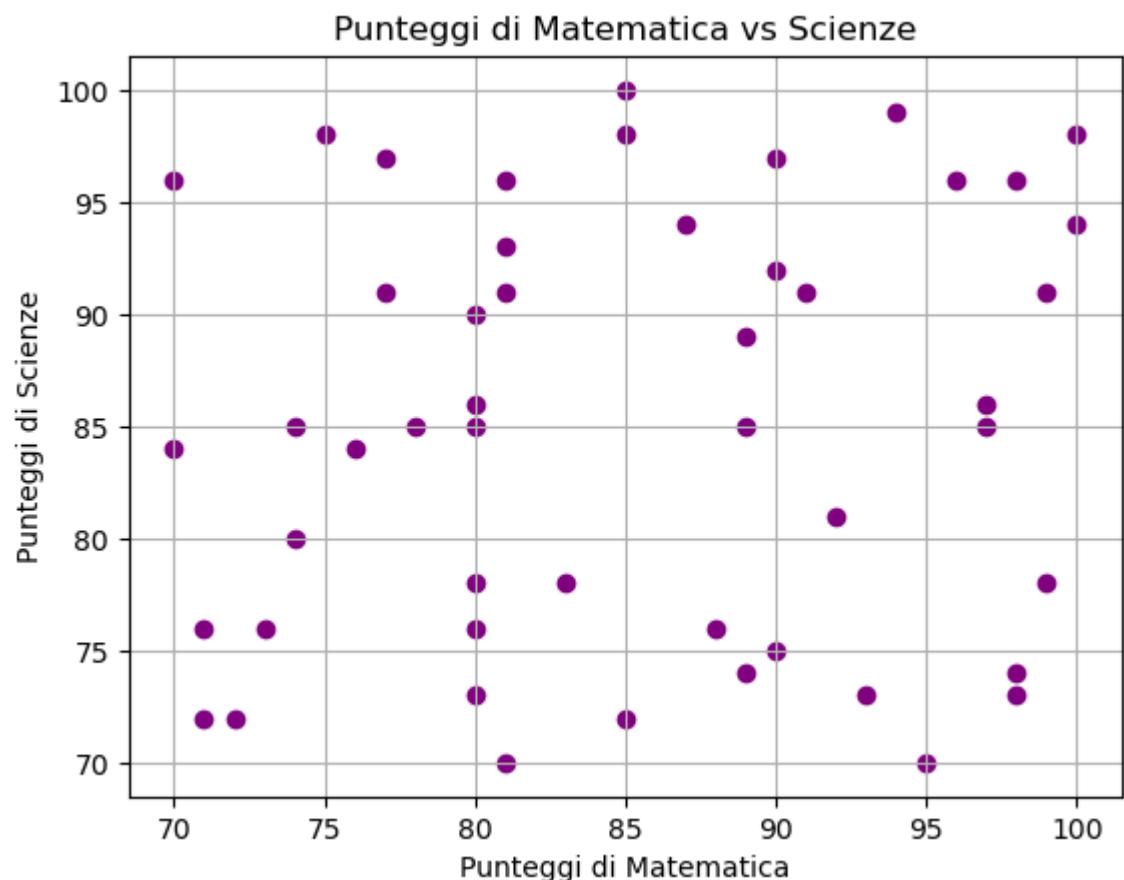
```
plt.scatter(punteggi_matematica, punteggi_scienze, color='purple', marker='o')
```

# Passo 4: Personalizza il grafico

```
plt.title('Punteggi di Matematica vs Scienze')
plt.xlabel('Punteggi di Matematica')
plt.ylabel('Punteggi di Scienze')
plt.grid(True)
```

# Passo 5: Mostra il grafico risultante

```
plt.show()
```



In [ ]:

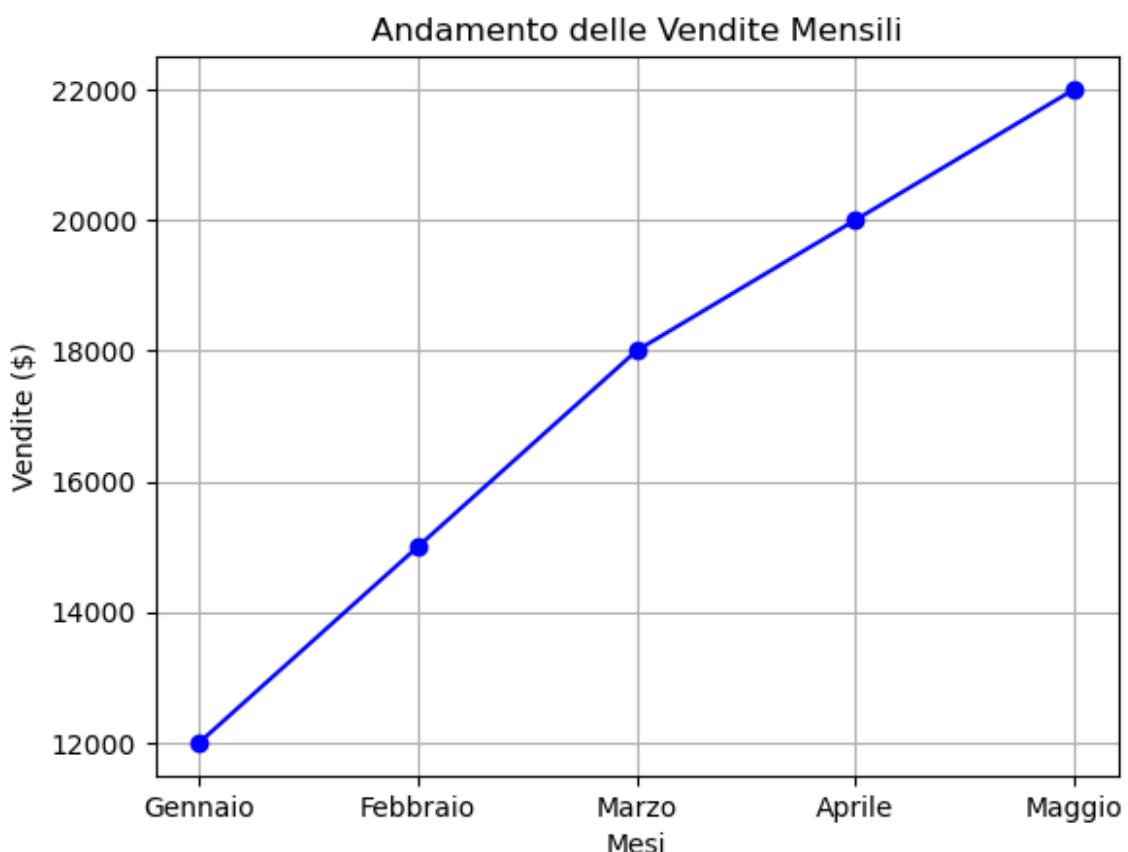
In [ ]:

In [121]:

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

```
mesi = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
vendite = [12000, 15000, 18000, 20000, 22000]
```

```
plt.plot(mesi, vendite, marker='o', linestyle='-', color='blue')
plt.title('Andamento delle Vendite Mensili')
plt.xlabel('Mesи')
plt.ylabel('Vendite ($)')
plt.grid(True)
plt.show()
```



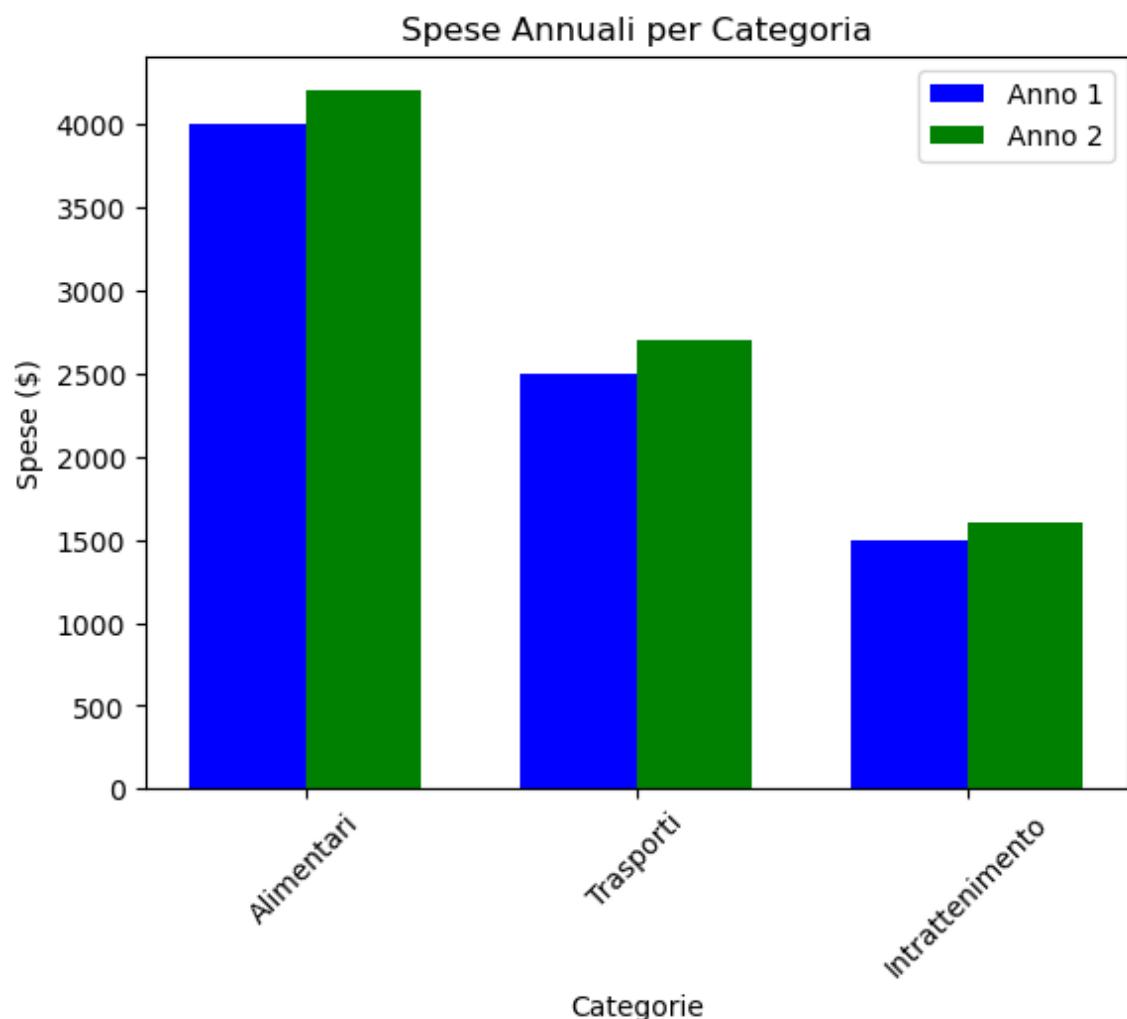
```
In [7]: import matplotlib.pyplot as plt
import numpy as np

categorie = ['Alimentari', 'Trasporti', 'Intrattenimento']
spese_anno_1 = [4000, 2500, 1500]
spese_anno_2 = [4200, 2700, 1600]

larghezza_barre = 0.35
indici = np.arange(len(categorie))

plt.bar(indici - larghezza_barre/2, spese_anno_1, width=larghezza_barre, label='Anno 1')
plt.bar(indici + larghezza_barre/2, spese_anno_2, width=larghezza_barre, label='Anno 2')

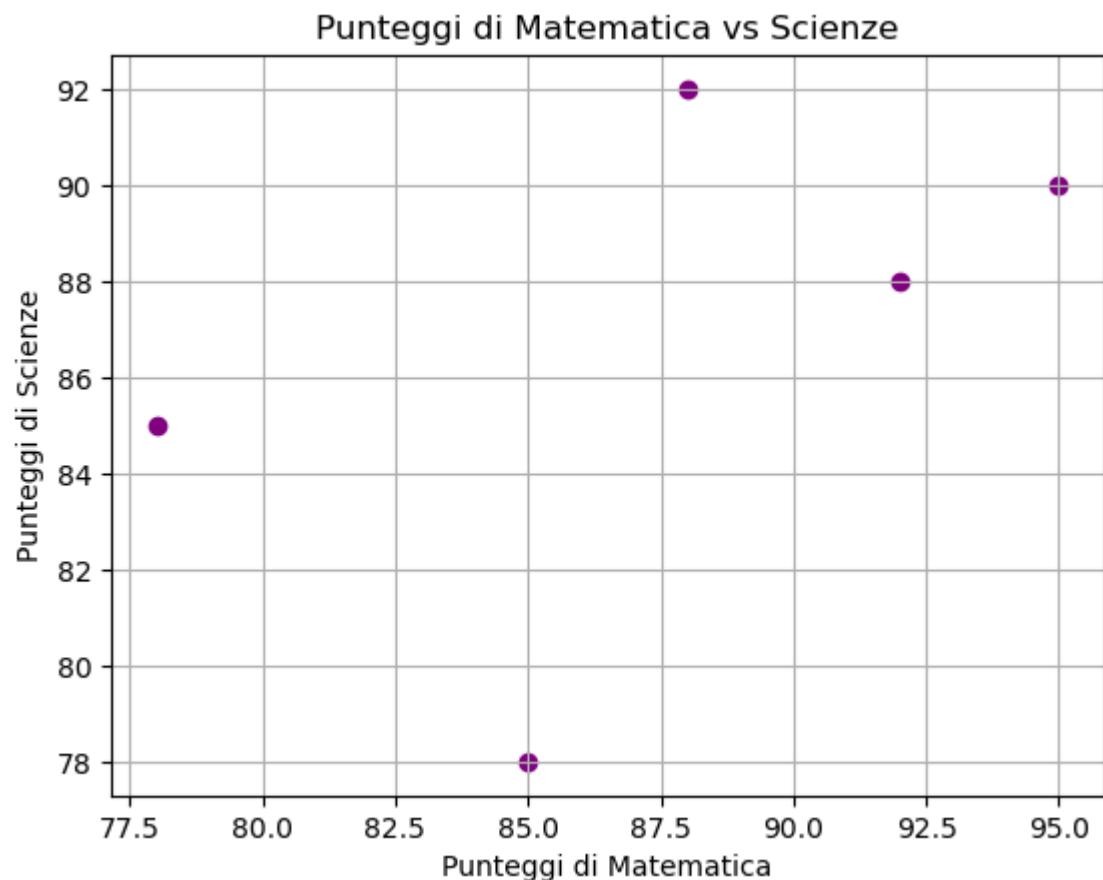
plt.title('Spese Annuali per Categoria')
plt.xlabel('Categorie')
plt.ylabel('Spese ($)')
plt.xticks(indici, categorie, rotation=45)
plt.legend(loc='upper right')
plt.show()
```



```
In [126]: import matplotlib.pyplot as plt
import numpy as np

punteggi_matematica = [85, 92, 78, 88, 95]
punteggi_scienze = [78, 88, 85, 92, 90]

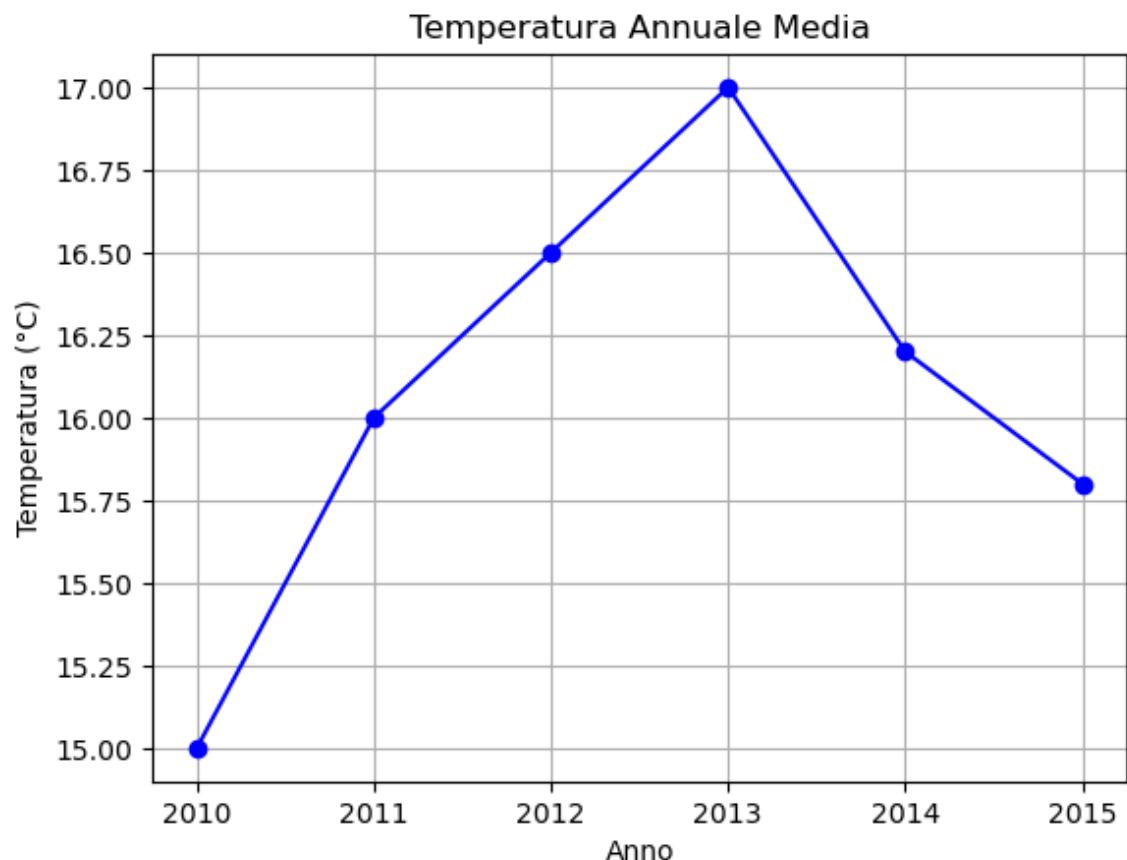
plt.scatter(punteggi_matematica, punteggi_scienze, color='purple', marker='o')
plt.title('Punteggi di Matematica vs Scienze')
plt.xlabel('Punteggi di Matematica')
plt.ylabel('Punteggi di Scienze')
plt.grid(True)
plt.show()
```



```
In [128]: import matplotlib.pyplot as plt

anni = [2010, 2011, 2012, 2013, 2014, 2015]
temperatura_media = [15, 16, 16.5, 17, 16.2, 15.8]

plt.plot(anni, temperatura_media, marker='o', linestyle='-', color='blue')
plt.title('Temperatura Annuale Media')
plt.xlabel('Anno')
plt.ylabel('Temperatura (°C)')
plt.grid(True)
plt.show()
```



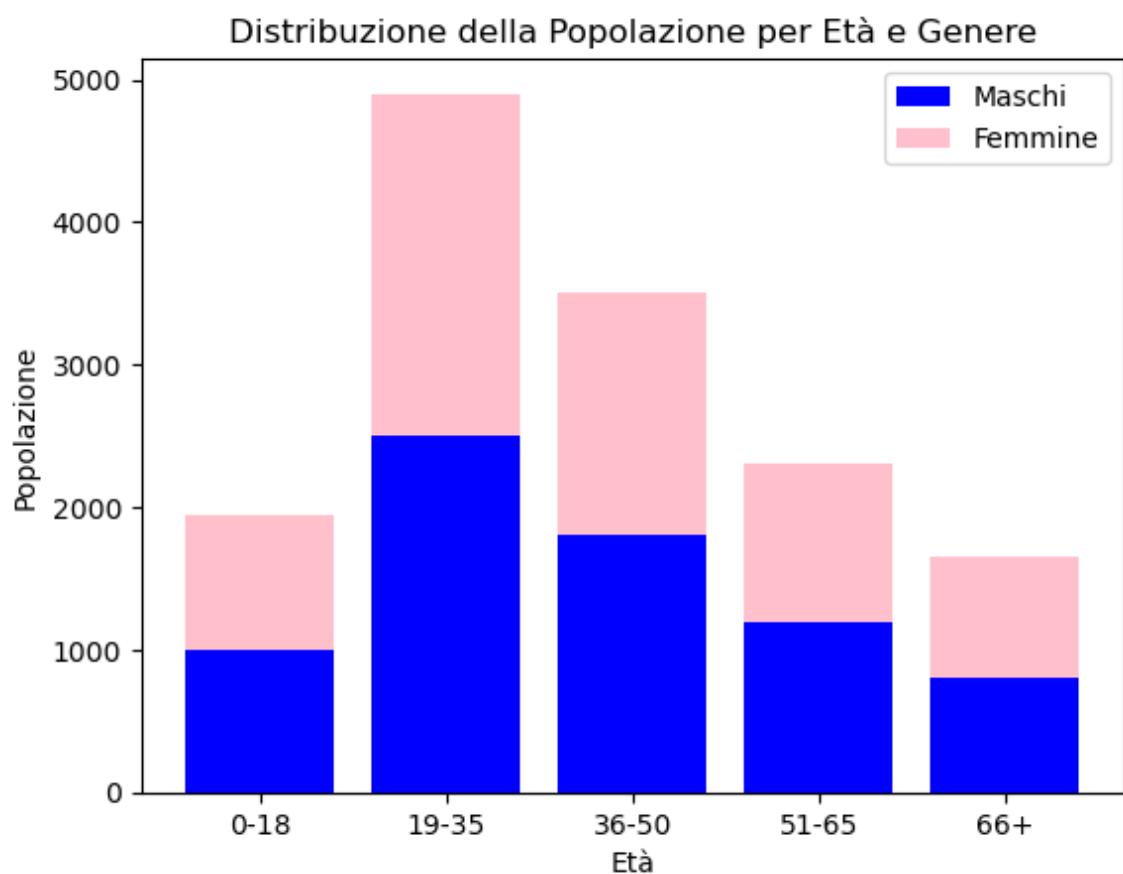
```
In [131]: import matplotlib.pyplot as plt
import numpy as np

età = ['0-18', '19-35', '36-50', '51-65', '66+']
popolazione_maschile = [1000, 2500, 1800, 1200, 800]
popolazione_femminile = [950, 2400, 1700, 1100, 850]

indici = np.arange(len(età))

plt.bar(età, popolazione_maschile, label='Maschi', color='blue')
plt.bar(età, popolazione_femminile, label='Femmine', bottom=popolazione_maschile)

plt.title('Distribuzione della Popolazione per Età e Genere')
plt.xlabel('Età')
plt.ylabel('Popolazione')
plt.legend(loc='upper right')
plt.show()
```



In [ ]:

In [ ]:

```
In [37]: import matplotlib.pyplot as plt
import numpy as np

fasce_eta = ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29', '30-34', '35-
popolazione_maschile = [1550000, 1600000, 1650000, 1700000, 1750000, 1780000]
popolazione_femminile = [1480000, 1550000, 1605000, 1650000, 1705000, 1750000]

indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

fig, ax1 = plt.subplots()

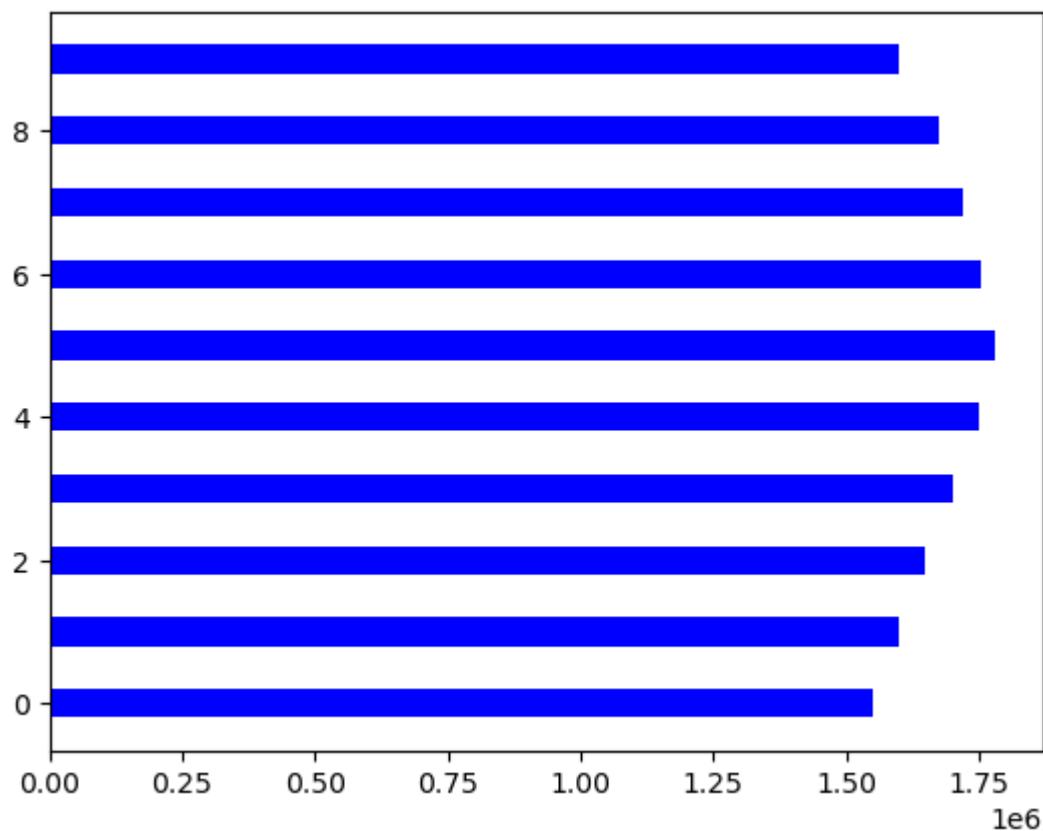
ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini')
ax1.barh(indici, -popolazione_femminile, height=larghezza_barre, label='Donne')

ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Sesso e Età')
ax1.set_yticks(indici)
ax1.set_yticklabels(fasce_eta)
ax1.legend(loc='upper right')
ax1.invert_yaxis() # Inverti l'asse y per rendere la piramide demografica

plt.show()
```

```
-----
-
TypeError                                 Traceback (most recent call last)
t)
~\AppData\Local\Temp\ipykernel_17604\1073391213.py in <module>
    12
    13     ax1.barh(indici, popolazione_maschile, height=larghezza_barre, lab
el='Uomini', color='blue')
--> 14     ax1.barh(indici, -popolazione_femminile, height=larghezza_barre, l
abel='Donne', color='pink')
    15
    16     ax1.set_xlabel('Popolazione')

TypeError: bad operand type for unary -: 'list'
```



```
In [1]: import matplotlib.pyplot as plt
import numpy as np

fasce_eta = ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29', '30-34', '35-39', '40-44', '45-49']
popolazione_maschile = [1550000, 1600000, 1650000, 1700000, 1750000, 1780000, 1800000, 1750000, 1700000, 1650000]
popolazione_femminile = [1480000, 1550000, 1605000, 1650000, 1705000, 1750000, 1800000, 1750000, 1700000, 1650000]

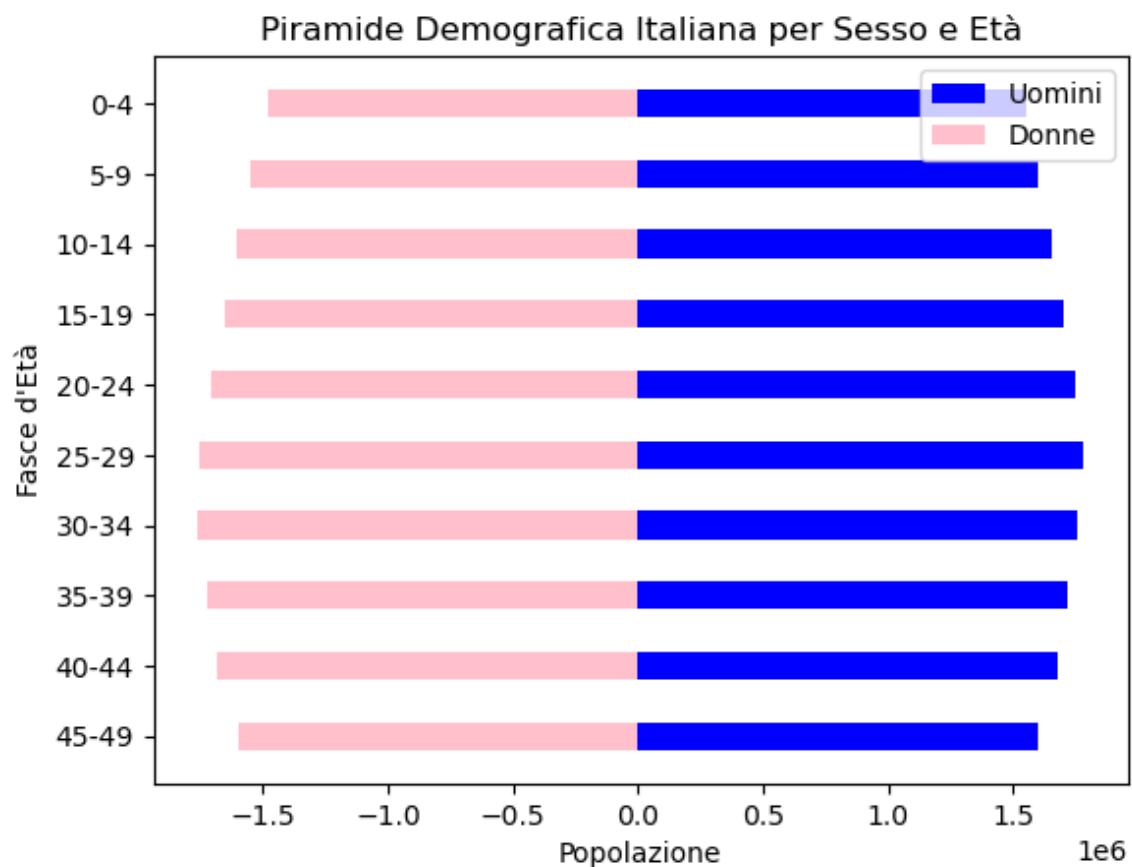
indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

fig, ax1 = plt.subplots()

ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini')
ax1.barh(indici, [-x for x in popolazione_femminile], height=larghezza_barre, label='Donne')

ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Sesso e Età')
ax1.set_yticks(indici)
ax1.set_yticklabels(fasce_eta)
ax1.legend(loc='upper right')
ax1.invert_yaxis() # Inverti l'asse y per rendere la piramide demografica

plt.show()
```



```
In [43]: import matplotlib.pyplot as plt
import numpy as np

fasce_eta = ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29', '30-34', '35-39', '40-44', '45-49']
popolazione_maschile = [100000, 110000, 120000, 130000, 140000, 130000, 120000, 110000, 100000, 90000]
popolazione_femminile = [95000, 105000, 115000, 125000, 135000, 125000, 115000, 105000, 95000, 85000]

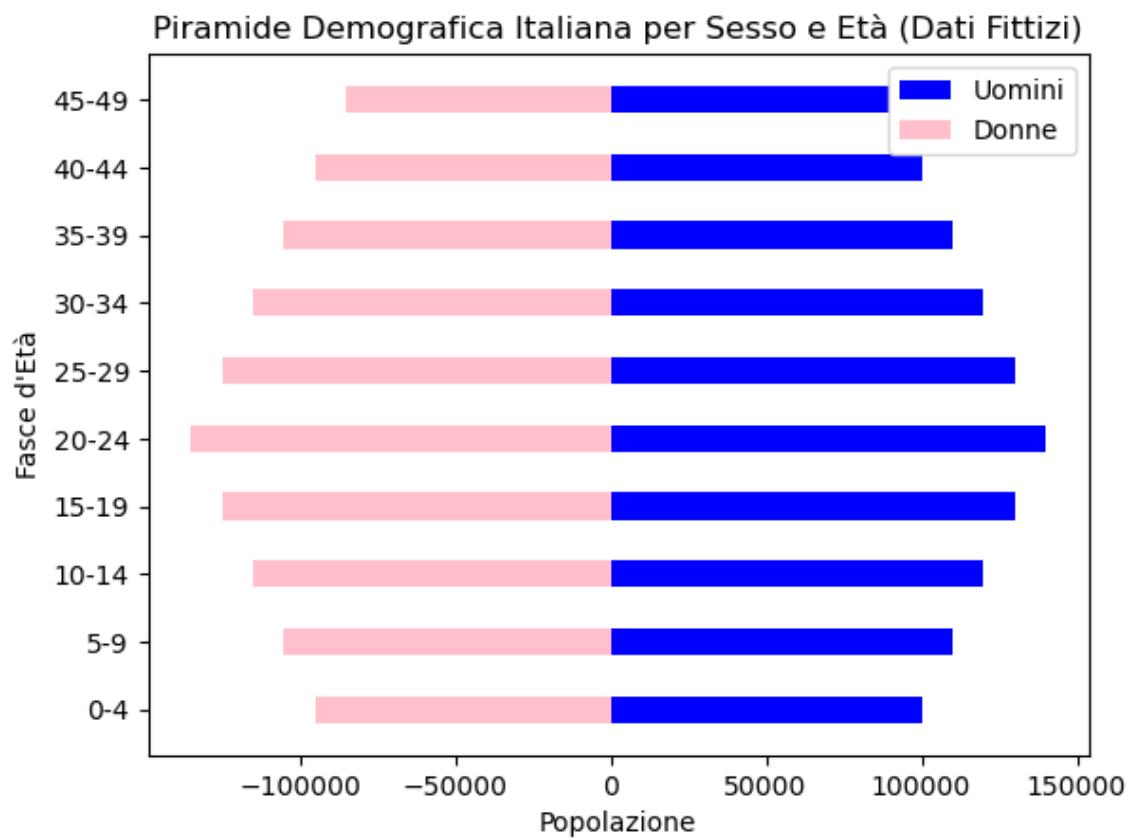
indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

fig, ax1 = plt.subplots()

ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini')
ax1.barh(indici, [-x for x in popolazione_femminile], height=larghezza_barre, label='Donne')

ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Sesso e Età (Dati Fittizi)')
ax1.set_yticks(indici)
ax1.set_yticklabels(fasce_eta)
ax1.legend(loc='upper right')
#ax1.invert_yaxis() # Inverte l'asse y per rendere la piramide demografica

plt.show()
```



```
In [50]: fasce_eta = ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29', '30-34', '35-39', '40-44', '45-49', '50-54', '55-59', '60-64', '65-69', '70-74', '75-79', '80-84', '85-89']

# Popolazione totale italiana (in milioni)
popolazione_totale = 60

# Stima realistica della popolazione maschile per ogni fascia di età (dati 1991)
percentuali_maschili = [49.5, 49.0, 48.5, 48.0, 47.5, 47.0, 46.5, 46.0, 45.5, 45.0, 44.5, 44.0, 43.5, 43.0, 42.5, 42.0, 41.5, 41.0, 40.5]

popolazione_maschile = []
popolazione_femminile = []

for percentuale in percentuali_maschili:
    popolazione_maschile.append(round((percentuale / 100) * (popolazione_totale / 1000000)))

# Calcolo della popolazione femminile per ogni fascia di età
for maschile in popolazione_maschile:
    popolazione_femminile = [(popolazione_totale * 1000000) - maschile]

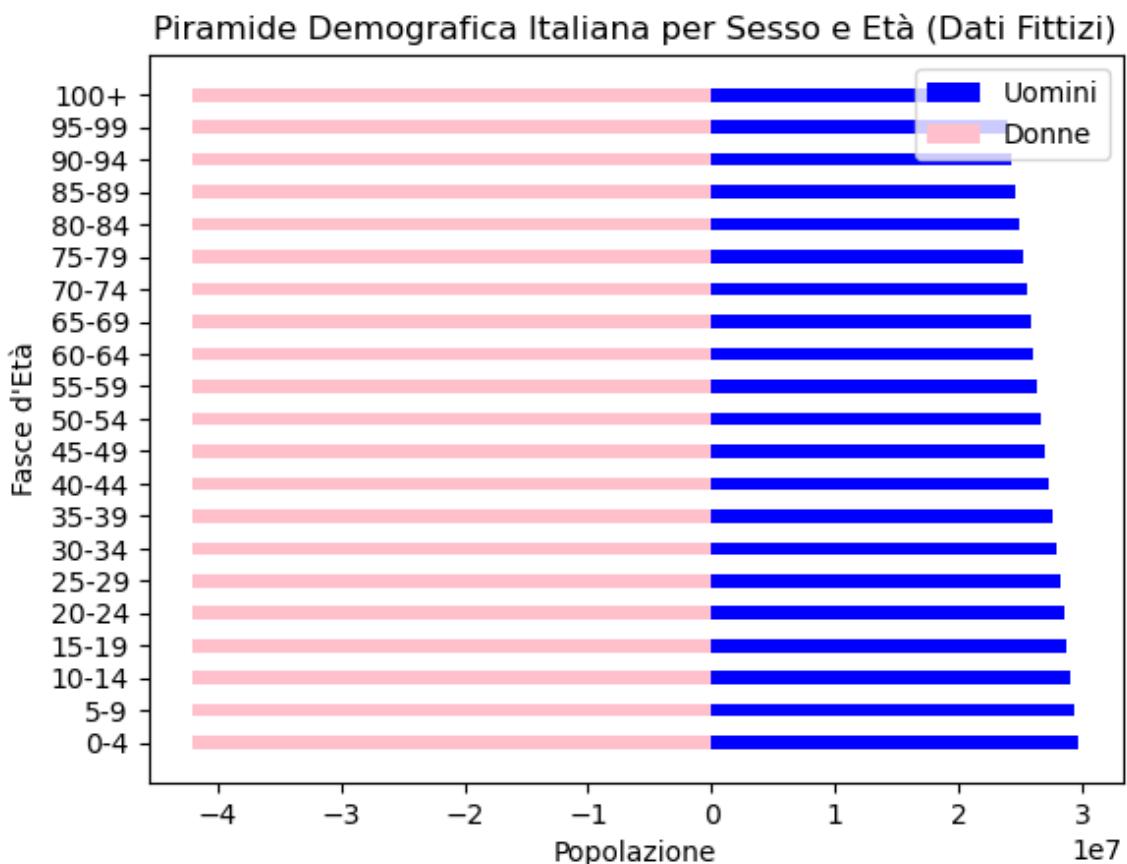
indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

fig, ax1 = plt.subplots()

ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini')
ax1.barh(indici, [-x for x in popolazione_femminile], height=larghezza_barre, label='Donne')

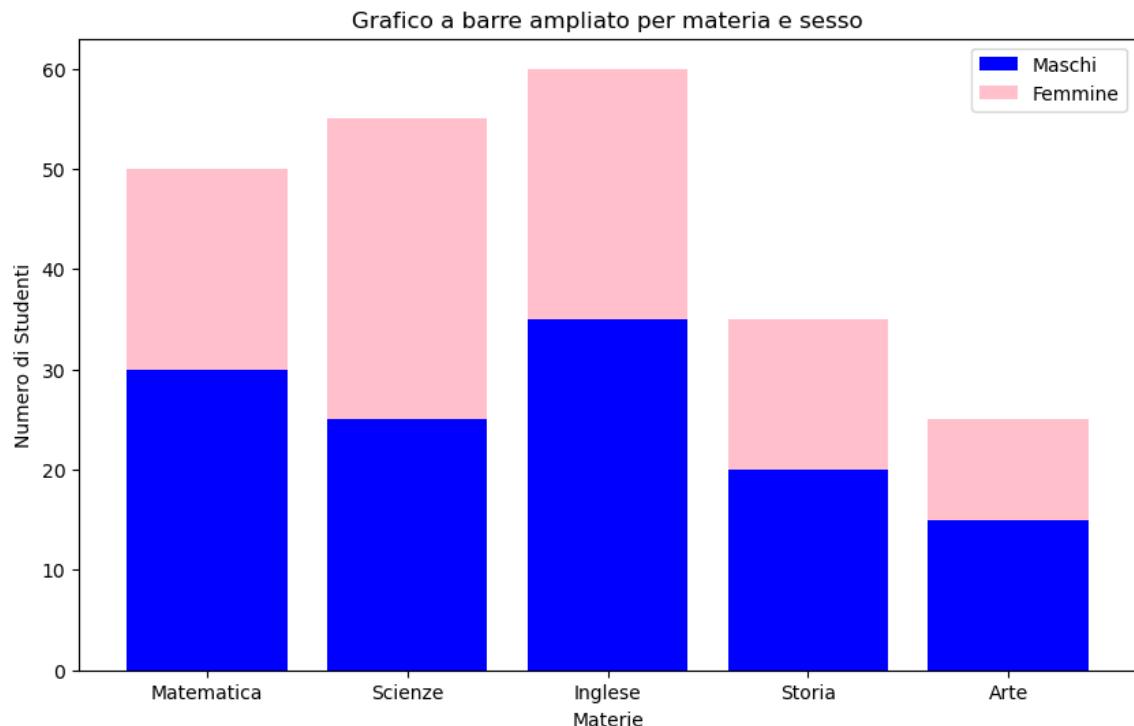
ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Sesso e Età (Dati Fittizi)')
ax1.set_yticks(indici)
ax1.set_yticklabels(fasce_eta)
ax1.legend(loc='upper right')
#ax1.invert_yaxis() # Inverte l'asse y per rendere la piramide demografica

plt.show()
```



In [6]: #Dati di esempio

```
materie = ['Matematica', 'Scienze', 'Inglese', 'Storia', 'Arte']
maschi = [30, 25, 35, 20, 15] # Numero di studenti maschi per materia
femmine = [20, 30, 25, 15, 10] # Numero di studentesse per materia
plt.figure(figsize=(10, 6))
plt.bar(materie, maschi, label='Maschi', color='blue')
plt.bar(materie, femmine, label='Femmine', bottom=maschi, color='pink')
plt.title('Grafico a barre ampliato per materia e sesso')
plt.xlabel('Materie')
plt.ylabel('Numero di Studenti')
plt.legend(loc='upper right')
plt.show()
```



In [8]: nome=input("dimmi come ti chiami")
for i in range(5):
 print("ciao",nome,"come stai?")

```
dimmi come ti chiamiAlessandro
ciao Alessandro come stai?
```

In [ ]:

```
In [1]: import pandas as pd

# Dataset con dati mancanti rappresentati da None o NaN
dataset = [
    {"età": 25, "punteggio": 90, "ammesso": 1},
    {"età": None, "punteggio": 85, "ammesso": 0},
    {"età": 28, "punteggio": None, "ammesso": 1},
    {"età": None, "punteggio": 75, "ammesso": 1},
    {"età": 23, "punteggio": None, "ammesso": None},
    {"età": 23, "punteggio": 77, "ammesso": None},
]
df = pd.DataFrame(dataset)
df
```

Out[1]:

	età	punteggio	ammesso
0	25.0	90.0	1.0
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

```
In [2]: df[["punteggio", "ammesso"]]
```

Out[2]:

	punteggio	ammesso
0	90.0	1.0
1	85.0	0.0
2	NaN	1.0
3	75.0	1.0
4	NaN	NaN
5	77.0	NaN

```
In [3]: # Identificazione delle righe con dati mancanti
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
righe_con_dati_mancanti
```

Out[3]:

	età	punteggio	ammesso
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

```
In [4]: # Conta quante righe con dati mancanti ci sono in totale
totale_dati_mancanti = righe_con_dati_mancanti.shape[0]
totale_dati_mancanti
```

Out[4]: 5

```
In [5]: print("Righe con dati mancanti:")
print(righe_con_dati_mancanti)
print("Totale dati mancanti:", totale_dati_mancanti)
```

Righe con dati mancanti:

	età	punteggio	ammesso
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

Totale dati mancanti: 5

```
In [6]: import pandas as pd
```

```
# Dataset con dati mancanti rappresentati da None o NaN
dataset = [
    {"nome": "Alice", "età": 25, "punteggio": 90, "email": "alice@email.com"},
    {"nome": "Bob", "età": 22, "punteggio": None, "email": None},
    {"nome": "Charlie", "età": 28, "punteggio": 75, "email": "charlie@email.com"}
]

# Converti il dataset in un DataFrame
df = pd.DataFrame(dataset)
df
```

Out[6]:

	nome	età	punteggio	email
0	Alice	25	90.0	alice@email.com
1	Bob	22	NaN	None
2	Charlie	28	75.0	charlie@email.com

```
In [7]: # Rimuovi le righe con dati mancanti
df1=df.dropna(inplace=False)
df1
```

Out[7]:

	nome	età	punteggio	email
0	Alice	25	90.0	alice@email.com
2	Charlie	28	75.0	charlie@email.com

```
In [8]: # Rimuovi le righe con dati mancanti
df.dropna(inplace=True)
df
```

Out[8]:

	nome	età	punteggio	email
0	Alice	25	90.0	alice@email.com
2	Charlie	28	75.0	charlie@email.com

In [ ]:

```
In [9]: import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Genera dati di esempio
data = {
    'Variable1': [1, 2, 3, 4, 5],
    'Variable2': [1, 2, np.nan, 4, np.nan],
    'Missing_Column': ['A', 'B', 'A', 'C', np.nan]
}
# Crea un DataFrame
df = pd.DataFrame(data)
df1=pd.DataFrame()
df
```

Out[9]:

	Variable1	Variable2	Missing_Column
0	1	1.0	A
1	2	2.0	B
2	3	NaN	A
3	4	4.0	C
4	5	NaN	NaN

In [10]: # Trattamento dei missing values nelle variabili numeriche  
 numeric\_cols = df.select\_dtypes(include=['number'])  
 numeric\_cols.columns

Out[10]: Index(['Variable1', 'Variable2'], dtype='object')

In [11]: df1[numeric\_cols.columns] = df[numeric\_cols.columns].fillna(df[numeric\_cols.columns].mean())  
 df1

Out[11]:

	Variable1	Variable2
0	1	1.000000
1	2	2.000000
2	3	2.333333
3	4	4.000000
4	5	2.333333

In [12]: # Trattamento dei missing values nelle variabili categoriche  
 categorical\_cols = df.select\_dtypes(exclude=['number'])  
 categorical\_cols.columns

Out[12]: Index(['Missing\_Column'], dtype='object')

In [13]: `df1[categorical_cols.columns] = df[categorical_cols.columns].fillna(df[categorical_cols.columns].mode().iloc[0])`

Out[13]:

	Variable1	Variable2	Missing_Column
0	1	1.000000	A
1	2	2.000000	B
2	3	2.333333	A
3	4	4.000000	C
4	5	2.333333	A

In [14]: `print(f"il primo con i valori mancanti \n{df} \ne il secondo con i missing \n{df1}")`

```
il primo con i valori mancanti
   Variable1  Variable2 Missing_Column
0            1          1.0              A
1            2          2.0              B
2            3          NaN              A
3            4          4.0              C
4            5          NaN            NaN
e il secondo con i missing values sostituiti
   Variable1  Variable2 Missing_Column
0            1          1.000000          A
1            2          2.000000          B
2            3          2.333333          A
3            4          4.000000          C
4            5          2.333333          A
```

In [1]:

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

# Genera dati di esempio
data = {
    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}
# Crea un DataFrame
df = pd.DataFrame(data)
df
```

Out[1]:

	Feature1	Feature2	Feature3
0	1.0	NaN	1.0
1	2.0	2.0	NaN
2	NaN	3.0	3.0
3	4.0	4.0	4.0
4	5.0	NaN	5.0

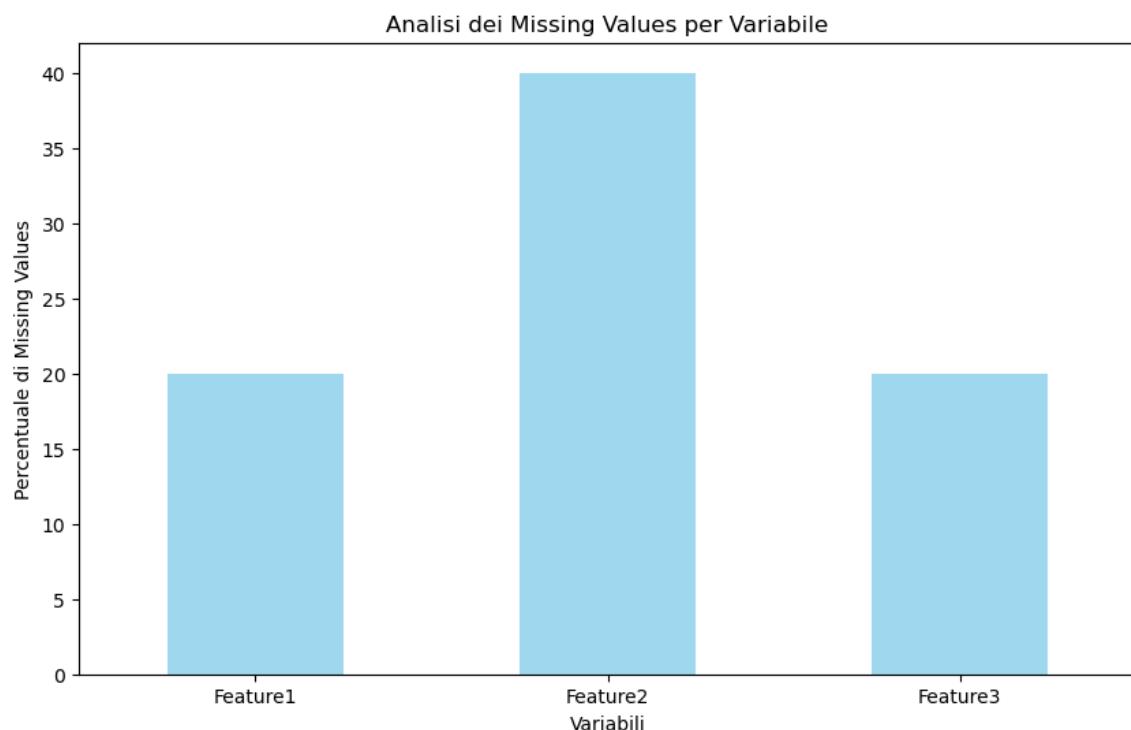
```
In [4]: df.isnull().sum()
```

```
Out[4]: Feature1      0.2
Feature2      0.4
Feature3      0.2
dtype: float64
```

```
In [5]: missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent
```

```
Out[5]: Feature1      20.0
Feature2      40.0
Feature3      20.0
dtype: float64
```

```
In [18]: # Calcola la percentuale di righe con missing values per ciascuna variabile
# Crea il grafico a barre
plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue', alpha=0.8)
plt.xlabel('Variabili')
plt.ylabel('Percentuale di Missing Values')
plt.title('Analisi dei Missing Values per Variabile')
plt.xticks(rotation=0)
plt.show()
```



```
In [19]: #creo una funzione che si occupa solo dei missing values
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Genera dati di esempio
data = {
    'Variable1': [1, 2, 3, 4, 5],
    'Variable2': [1, 2, np.nan, 4, np.nan],
    'Missing_Column': ['A', 'B', 'A', 'C', np.nan]
}

# Crea un DataFrame
df = pd.DataFrame(data)
df1=pd.DataFrame()

def missingvalues_sub(df):
    # Trattamento dei missing values nelle variabili numeriche e categoriche
    numeric_cols = df.select_dtypes(include=['number'])
    categorical_cols = df.select_dtypes(exclude=['number'])
    df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.columns].mean())
    df1[categorical_cols.columns] = df[categorical_cols.columns].fillna(df[categorical_cols.columns].mode())
    return df1

def main():
    df1=missingvalues_sub(df)
    print(f"il primo con i valori mancanti \n{df} \ne il secondo con i missing values sostituiti \n{df1}")

if __name__ == "__main__":
    main()
```

```
il primo con i valori mancanti
  Variable1  Variable2 Missing_Column
0         1         1.0                  A
1         2         2.0                  B
2         3        NaN                  A
3         4         4.0                  C
4         5        NaN                  NaN
e il secondo con i missing values sostituiti
  Variable1  Variable2 Missing_Column
0         1  1.000000                  A
1         2  2.000000                  B
2         3  2.333333                  A
3         4  4.000000                  C
4         5  2.333333                  A
```

In [20]: `df.isnull()`

Out[20]:

	Variable1	Variable2	Missing_Column
0	False	False	False
1	False	False	False
2	False	True	False
3	False	False	False
4	False	True	True

In [ ]:

```
In [5]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Genera dati di esempio
data = {
    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}

# Crea un DataFrame
df = pd.DataFrame(data)

# Calcola la matrice di missing values
missing_matrix = df.isnull()
missing_matrix
```

Out[5]:

	Feature1	Feature2	Feature3
0	False	True	False
1	False	False	True
2	True	False	False
3	False	False	False
4	False	True	False

```
In [8]: # Crea una heatmap colorata  
plt.figure(figsize=(8, 6))  
sns.heatmap(missing_matrix, cmap='viridis', cbar=False, alpha=0.8)  
plt.title('Matrice di Missing Values')  
plt.show()
```



```
In [7]: #pip install plotly
```

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Genera dati casuali per l'+ 
np.random.seed(2)
data = {
    'Età': np.random.randint(18, 70, size=1000),
    'Genere': np.random.choice(['Maschio', 'Femmina'], size=1000),
    'Punteggio': np.random.uniform(0, 100, size=1000),
    'Reddito': np.random.normal(50000, 15000, size=1000)
}

df = pd.DataFrame(data)

# Visualizza le prime righe del dataset
print(df.head(37))
```

	Età	Genere	Punteggio	Reddito
0	58	Maschio	93.309731	55174.034340
1	33	Femmina	97.279382	65873.059029
2	63	Femmina	91.185842	63246.553249
3	26	Femmina	75.926276	44534.875858
4	40	Maschio	25.156395	73444.267270
5	61	Femmina	90.055564	48451.939402
6	36	Femmina	29.717079	44579.517216
7	29	Femmina	87.762886	74639.606864
8	58	Femmina	4.139801	84279.892767
9	25	Femmina	5.641115	52083.863707
10	52	Maschio	80.315899	58188.649042
11	67	Maschio	10.670863	40301.012748
12	49	Maschio	43.920719	58292.619116
13	29	Femmina	34.315554	54842.947703
14	39	Maschio	27.790752	53270.120207
15	65	Maschio	36.205126	78821.228153
16	49	Maschio	48.566180	59639.075018
17	44	Femmina	83.643168	39339.223303
18	38	Maschio	61.718371	40687.283872
19	55	Femmina	90.736827	66795.123408
20	57	Maschio	83.670954	66695.930851
21	21	Femmina	14.613108	13662.713756
22	56	Maschio	42.276431	43888.196970
23	22	Maschio	53.301312	50533.098959
24	60	Femmina	25.575777	37855.373132
25	61	Maschio	56.628623	30927.633574
26	69	Maschio	97.422086	37713.280607
27	57	Femmina	22.522042	63240.818026
28	56	Femmina	78.456278	67359.749115
29	60	Maschio	99.494878	62275.706472
30	51	Femmina	25.405670	42254.194544
31	21	Maschio	85.606149	68879.204124
32	23	Femmina	17.390205	60142.897684
33	42	Femmina	92.200775	59981.424290
34	22	Femmina	30.421383	57726.903107
35	64	Femmina	27.801032	17644.668081
36	24	Femmina	90.473494	46518.121943

```
In [3]: # Informazioni sul dataset
```

```
print(df.info())
```

```
# Statistiche descrittive
```

```
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Età         1000 non-null   int32  
 1   Genere      1000 non-null   object  
 2   Punteggio   1000 non-null   float64 
 3   Reddito     1000 non-null   float64 
dtypes: float64(2), int32(1), object(1)
memory usage: 27.5+ KB
None
Età      Punteggio      Reddito      
count   1000.000000  1000.000000  1000.000000
mean    44.205000   48.687071   50036.084395
std     14.986847   29.617200   15027.142896
min    18.000000   0.090182   6017.070033
25%    31.000000   22.373740   39577.758808
50%    44.000000   47.030664   50994.854630
75%    58.000000   75.439618   60933.234680
max    69.000000   99.713537   96435.848804
```

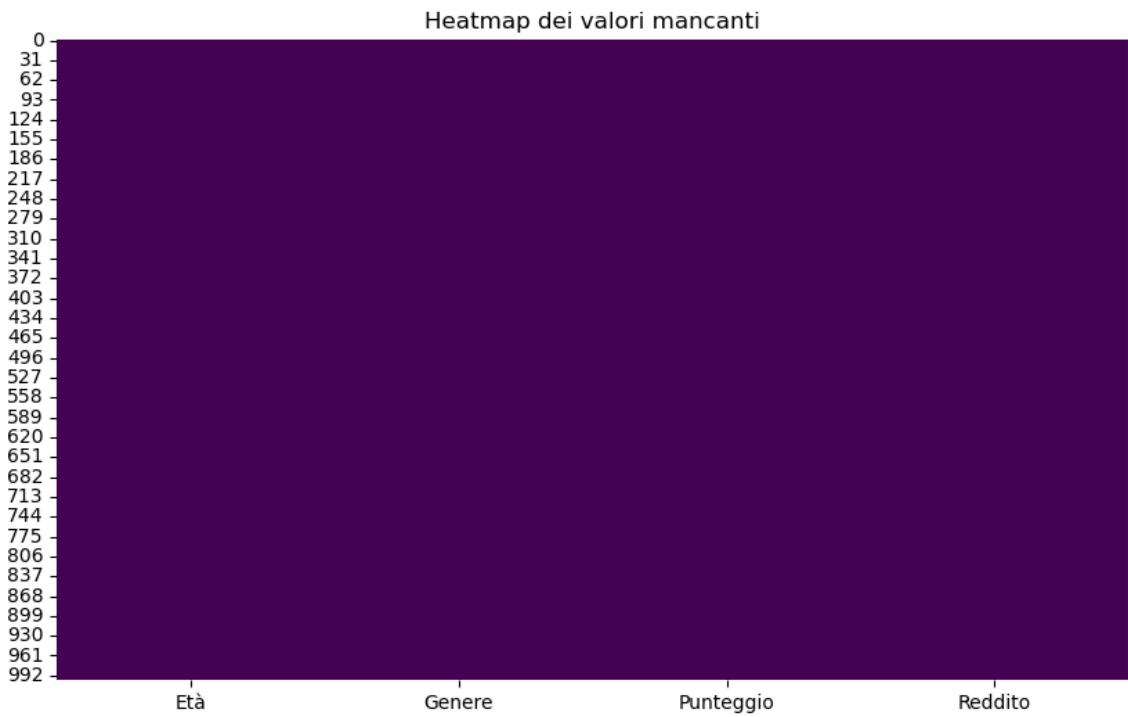
```
In [4]: # Gestione dei valori mancanti
```

```
missing_data = df.isnull().sum()
print("Valori mancanti per ciascuna colonna:")
print(missing_data)
```

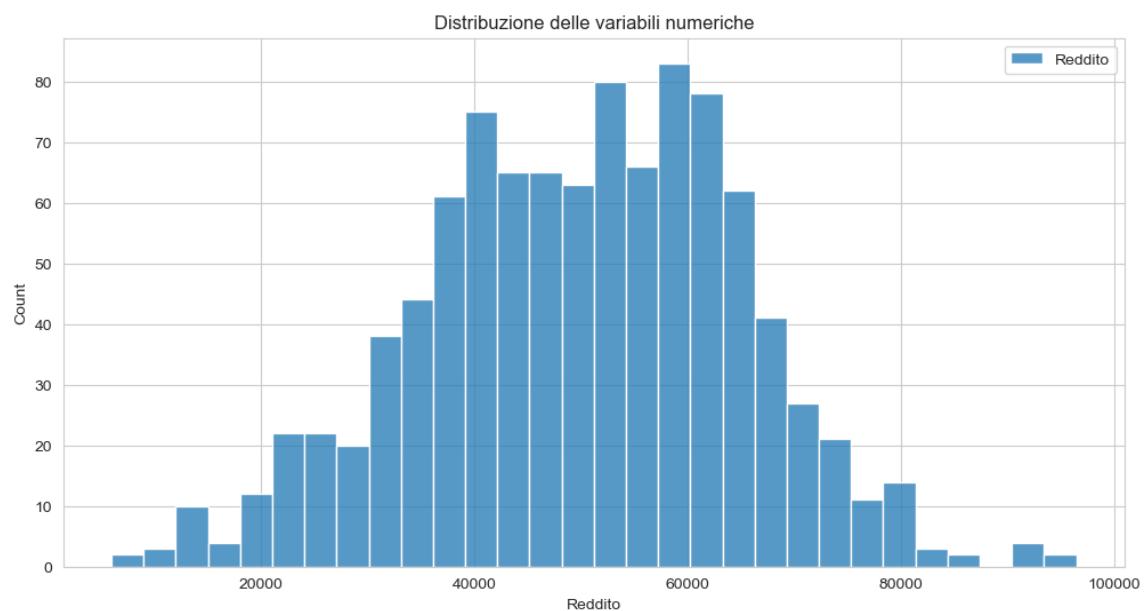
```
Valori mancanti per ciascuna colonna:
```

```
Età        0
Genere    0
Punteggio 0
Reddito    0
dtype: int64
```

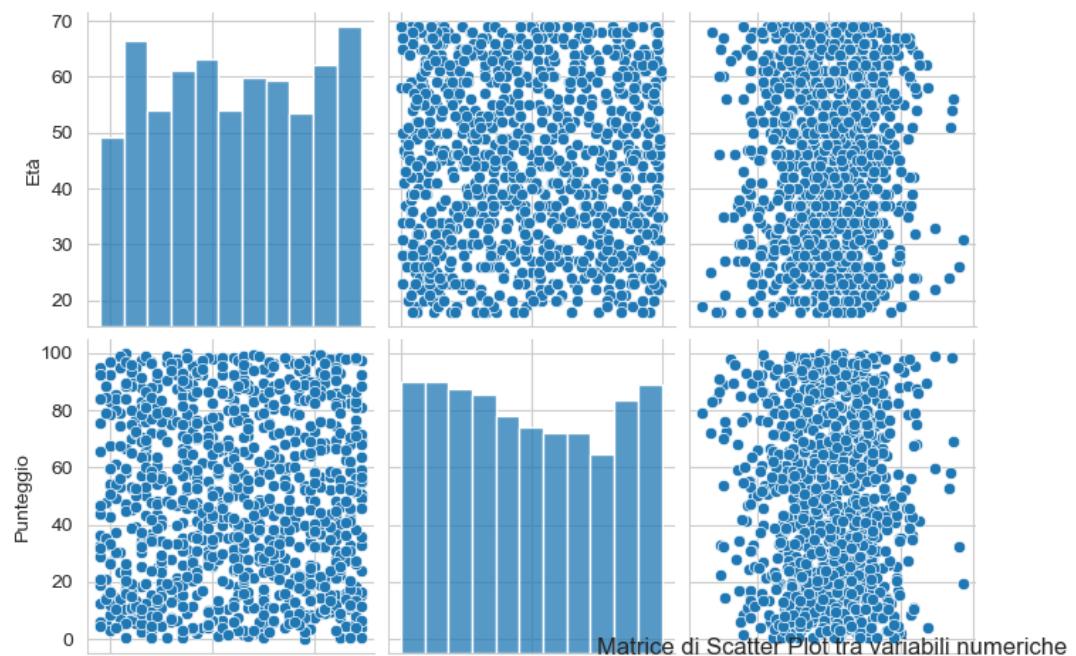
```
In [4]: # Visualizza una heatmap dei valori mancanti
missing_matrix=df.isnull()
plt.figure(figsize=(10, 6))
sns.heatmap(missing_matrix, cbar=False, cmap='viridis')
plt.title('Heatmap dei valori mancanti')
plt.show()
```



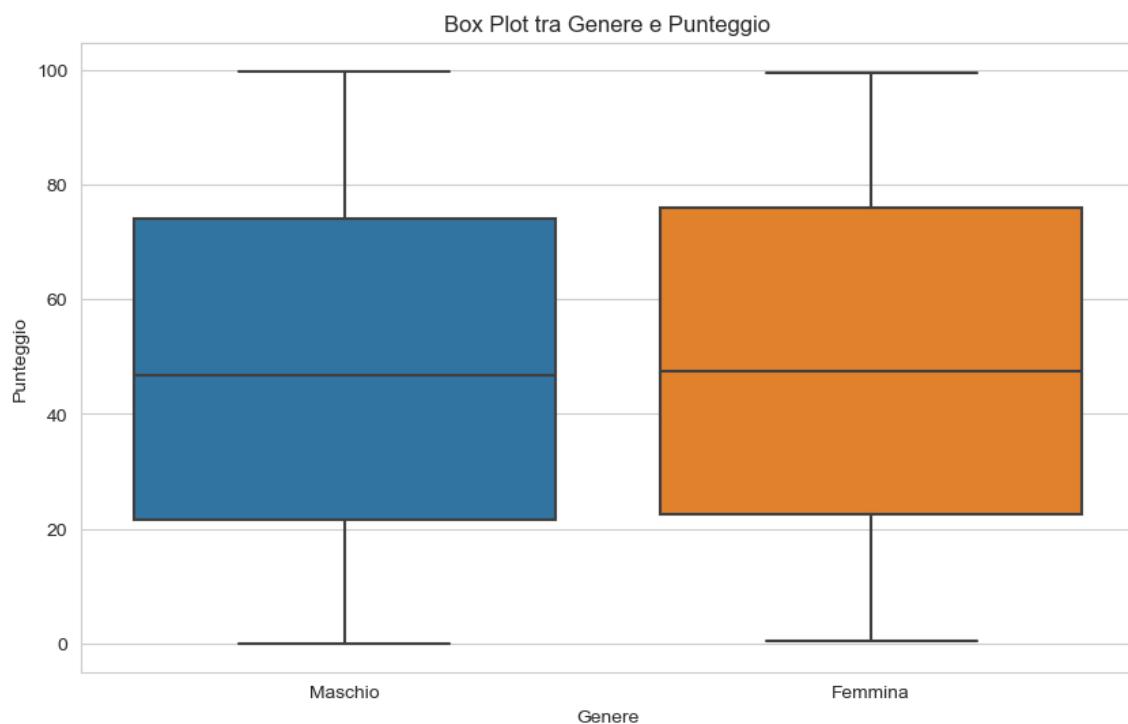
```
In [6]: # Visualizza la distribuzione delle variabili numeriche
plt.figure(figsize=(12, 6))
sns.set_style("whitegrid")
sns.histplot(df["Reddito"], kde=False, bins=30, label="Reddito")
plt.legend()
plt.title('Distribuzione delle variabili numeriche')
plt.show()
```



```
In [7]: # Visualizza una matrice di scatter plot tra le variabili numeriche  
numeric_features = df.select_dtypes(include=[np.number])  
sns.pairplot(df[numeric_features.columns])  
plt.title('Matrice di Scatter Plot tra variabili numeriche')  
plt.show()
```



```
In [8]: # Visualizza una box plot per una variabile numerica rispetto a una categoria  
plt.figure(figsize=(10, 6))  
sns.boxplot(x='Genere', y='Punteggio', data=df)  
plt.title('Box Plot tra Genere e Punteggio')  
plt.show()
```



```
In [ ]:
```

```
In [8]: # Visualizza un grafico a dispersione interattivo utilizzando Plotly
import plotly.express as px
fig = px.scatter(df, x='Età', y='Reddito', color='Genere', size='Punteggio')
fig.update_layout(title='Grafico a dispersione interattivo')
fig.show()
```

```
In [32]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Genera dati casuali per l'esplorazione
np.random.seed(22)
data = {
    'Data': pd.date_range(start='2023-01-01', end='2023-12-31', freq='D'),
    'Vendite': np.random.randint(100, 1000, size=365),
    'Prodotto': np.random.choice(['A', 'B', 'C'], size=365)
}

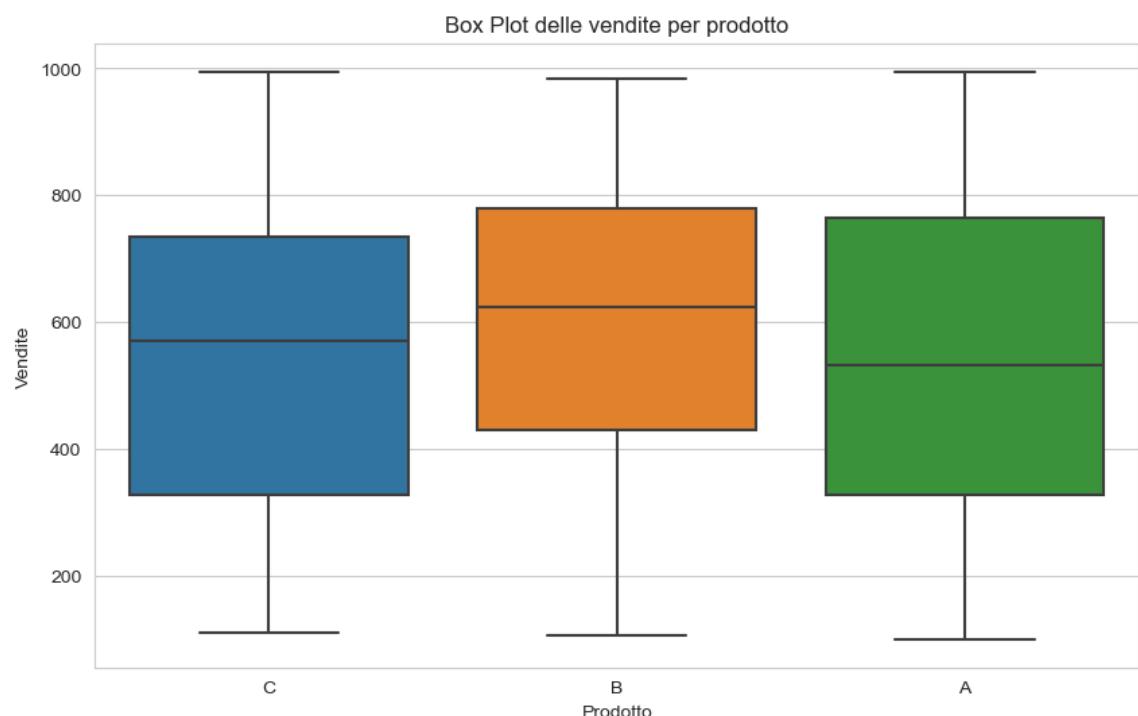
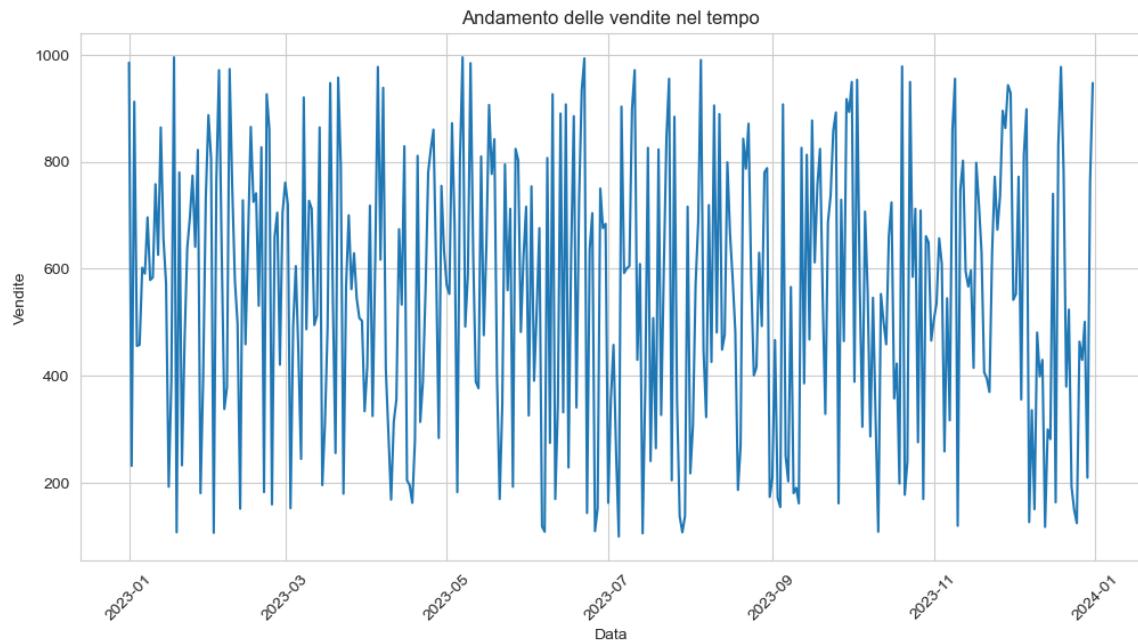
df = pd.DataFrame(data)

# Visualizza le prime righe del dataset
print(df.head())

# Visualizza un grafico delle vendite nel tempo
plt.figure(figsize=(12, 6))
sns.lineplot(x='Data', y='Vendite', data=df)
plt.title('Andamento delle vendite nel tempo')
plt.xlabel('Data')
plt.ylabel('Vendite')
plt.xticks(rotation=45)
plt.show()

# Visualizza una box plot delle vendite per prodotto
plt.figure(figsize=(10, 6))
sns.boxplot(x='Prodotto', y='Vendite', data=df)
plt.title('Box Plot delle vendite per prodotto')
plt.xlabel('Prodotto')
plt.ylabel('Vendite')
plt.show()
```

	Data	Vendite	Prodotto
0	2023-01-01	985	C
1	2023-01-02	232	B
2	2023-01-03	912	C
3	2023-01-04	456	B
4	2023-01-05	458	A



In [ ]:

```
In [14]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Genera dati di esempio
data = {
    'Numeric_Var': [1, 2, 3, 4, np.nan, 6],
    'Categorical_Var': ['A', 'B', 'A', 'B', 'A', 'B']
}

# Crea un DataFrame
df = pd.DataFrame(data)
print(df)
```

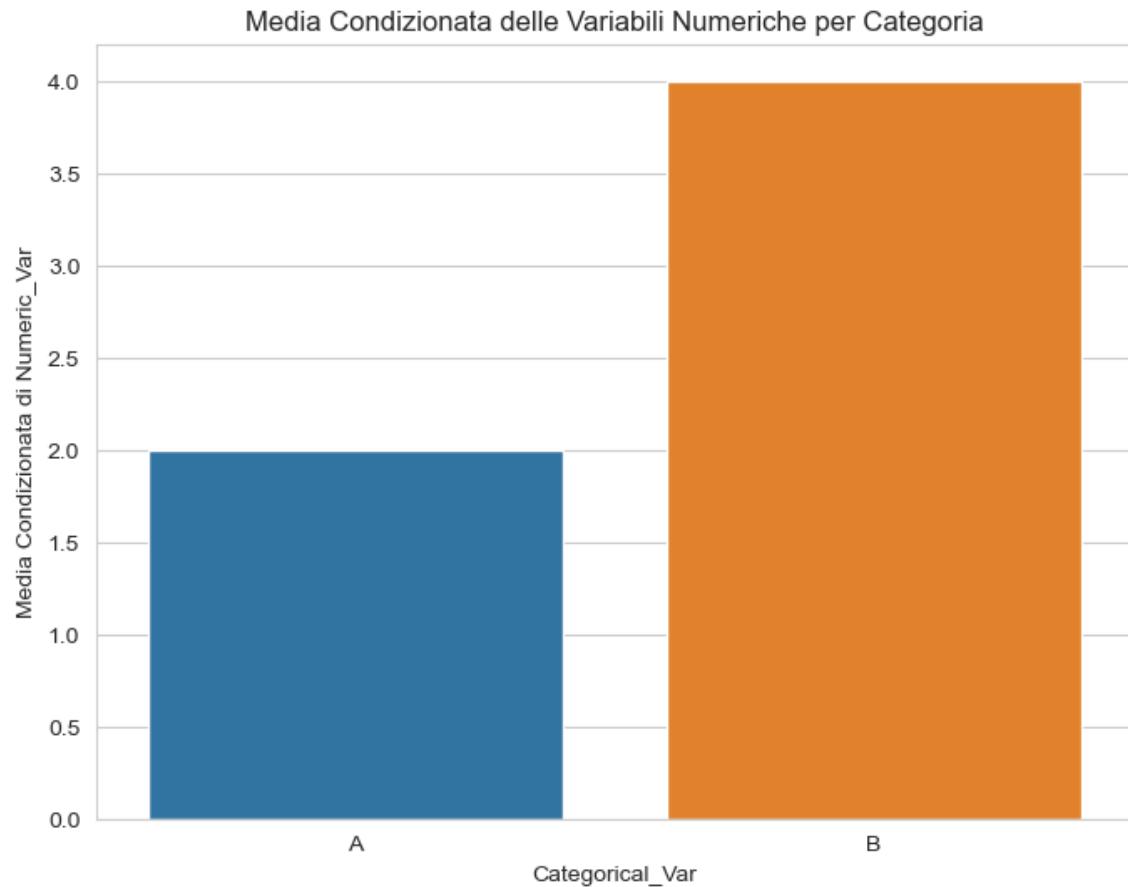
	Numeric_Var	Categorical_Var
0	1.0	A
1	2.0	B
2	3.0	A
3	4.0	B
4	NaN	A
5	6.0	B

```
In [16]: # Calcola la media condizionata
conditional_means = df['Numeric_Var'].fillna(df.groupby('Categorical_Var')[
```

```
In [34]: # Aggiorna la colonna Numeric_Var con la media condizionata
df['Numeric_Var'] = conditional_means
print(df)

# Crea un grafico a barre per mostrare la media condizionata per ogni categoria
plt.figure(figsize=(8, 6))
sns.barplot(data=df, x='Categorical_Var', y='Numeric_Var', ci=False)
plt.xlabel('Categorical_Var')
plt.ylabel('Media Condizionata di Numeric_Var')
plt.title('Media Condizionata delle Variabili Numeriche per Categoria')
plt.show()
```

	Numeric_Var	Categorical_Var
0	1.0	A
1	2.0	B
2	3.0	A
3	4.0	B
4	2.0	A
5	6.0	B



```
In [ ]:
```

```
In [36]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Genera dati casuali per l'esplorazione
np.random.seed(42)
data = {
    'Età': np.random.randint(18, 65, size=500),
    'Soddisfazione':
        np.random.choice(['Molto Soddisfatto', 'Soddisfatto', 'Neutro', 'Insoddisfatto'],
                        p=[0.2, 0.3, 0.3, 0.2])
}

df = pd.DataFrame(data)
print(df)
conditional_means = df.groupby('Soddisfazione')['Età'].transform('mean')

df['Numeric_Var'] = conditional_means
print(df)

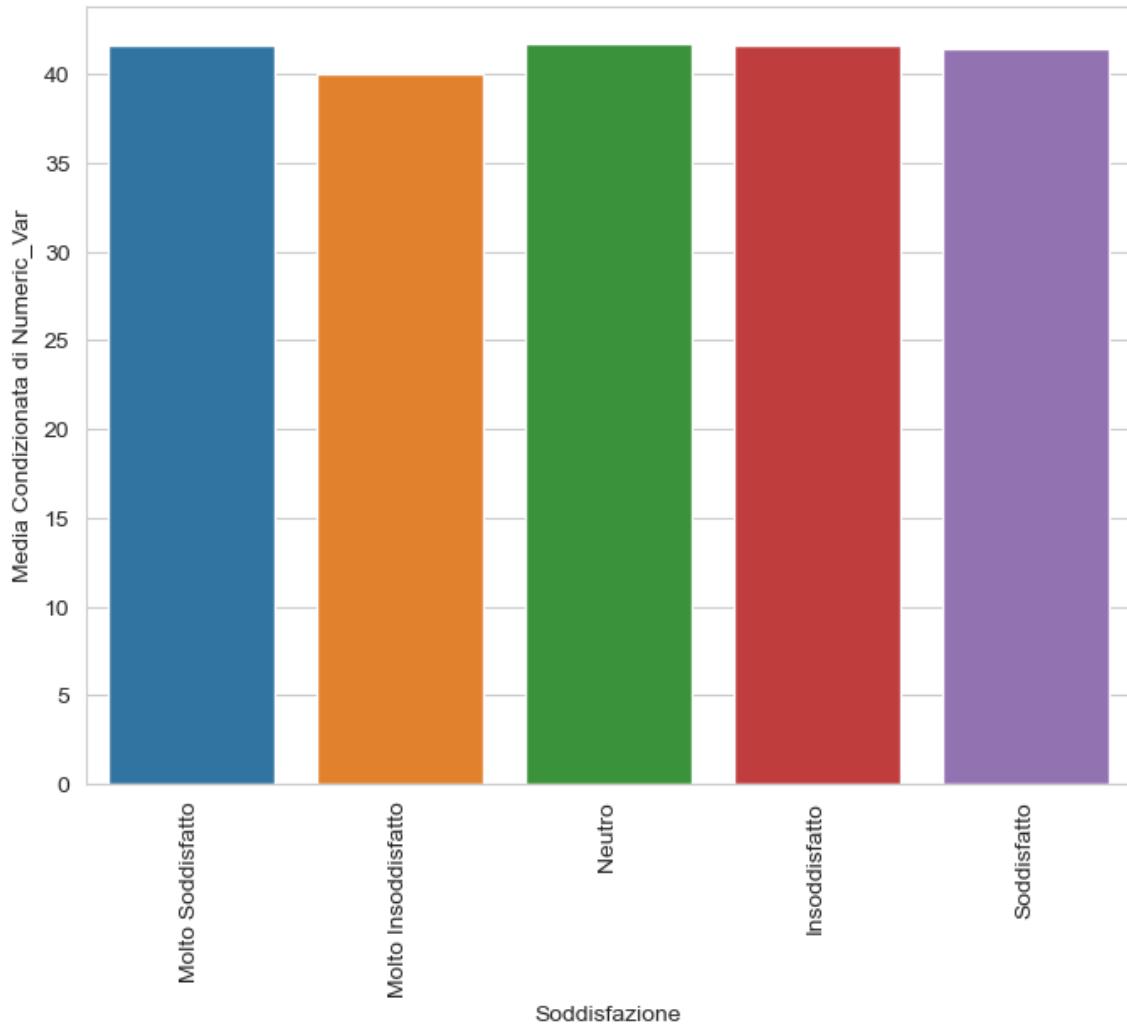
# Crea un grafico a barre per mostrare La media condizionata per ogni categoria
plt.figure(figsize=(8, 6))
sns.barplot(data=df, x='Soddisfazione', y='Numeric_Var', ci=None)
plt.xlabel('Soddisfazione')
plt.ylabel('Media Condizionata di Numeric_Var')
plt.title('Media Condizionata delle Variabili Numeriche per CATEGORIA')
plt.xticks(rotation=90)

plt.show()
```

	Età	Soddisfazione	Numeric_Var
0	56	Molto Soddisfatto	41.651376
1	46	Molto Insoddisfatto	40.054054
2	32	Neutro	41.747368
3	60	Neutro	41.747368
4	25	Molto Insoddisfatto	40.054054
..	...	...	...
495	37	Molto Soddisfatto	41.651376
496	41	Molto Soddisfatto	41.651376
497	29	Molto Soddisfatto	41.651376
498	52	Molto Soddisfatto	41.651376
499	50	Molto Soddisfatto	41.651376

[500 rows x 3 columns]

## Media Condizionata delle Variabili Numeriche per Categoria

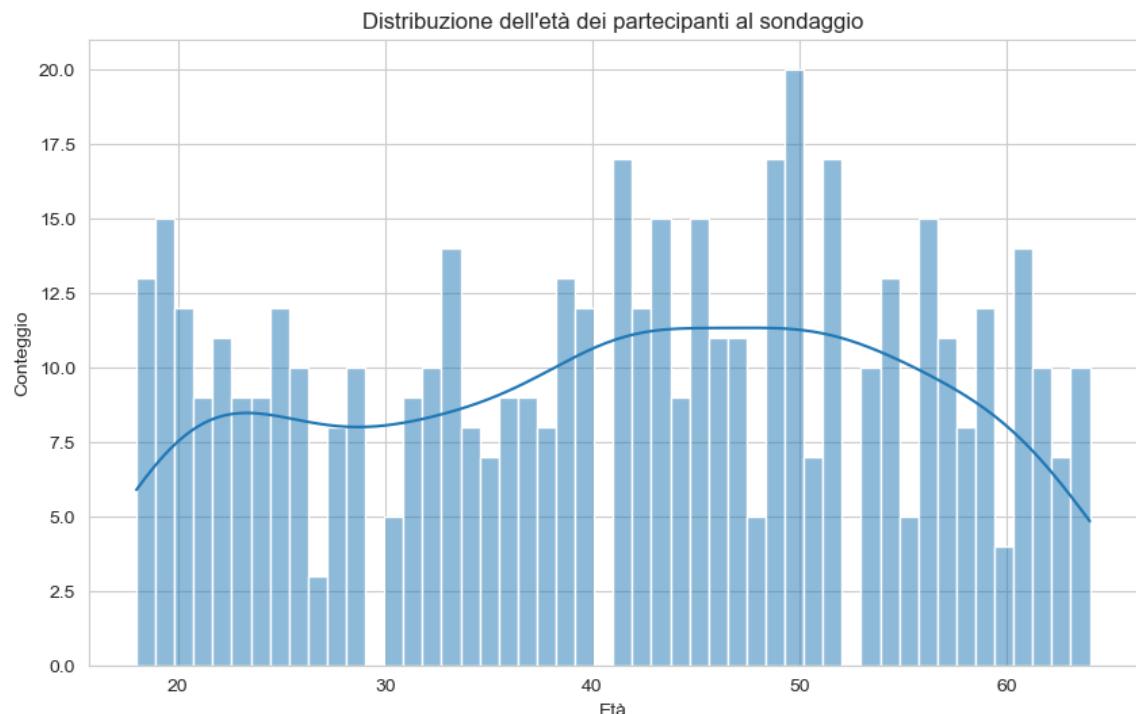


```
In [37]: # Visualizza le prime righe del dataset
print(df.head())

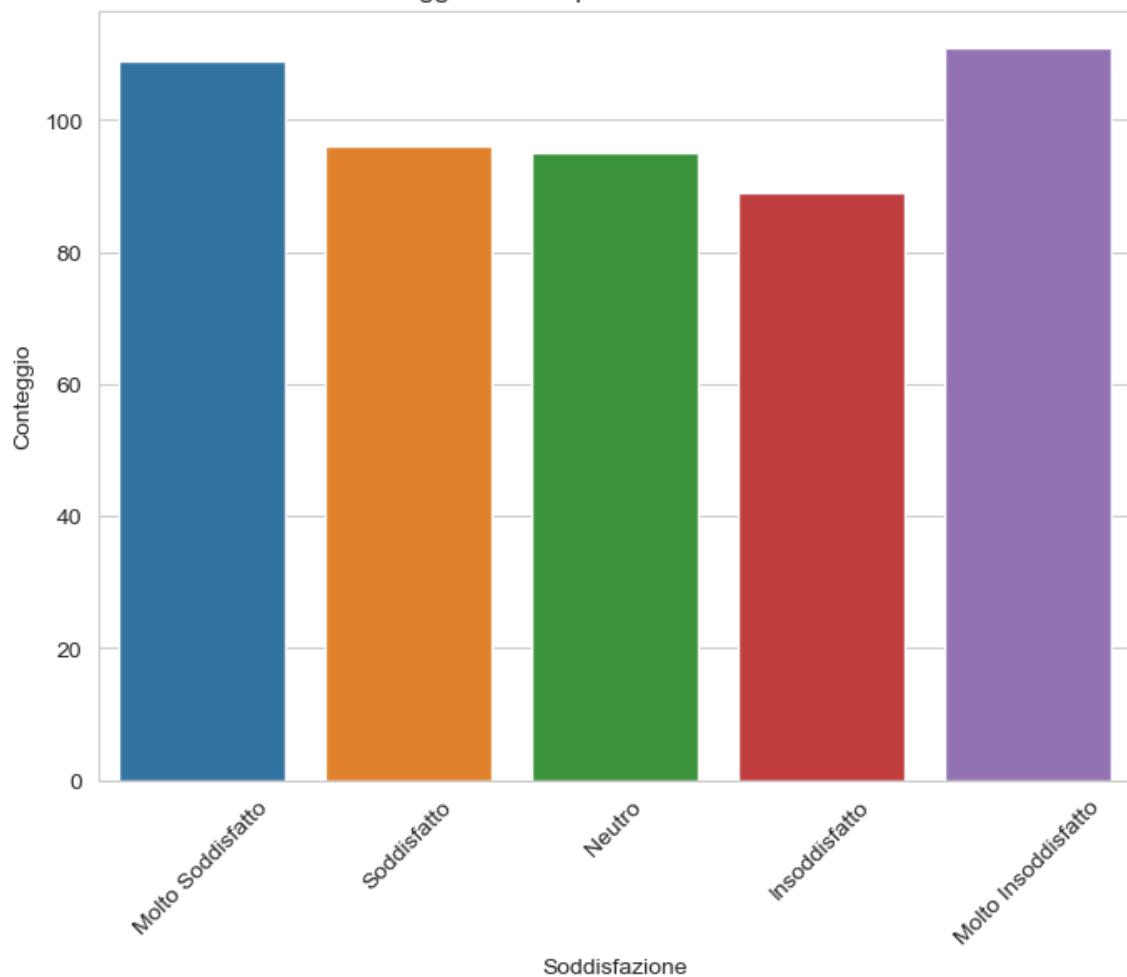
# Visualizza una distribuzione dell'età
plt.figure(figsize=(10, 6))
sns.histplot(df['Età'], bins=50, kde=True)
plt.title('Distribuzione dell\'età dei partecipanti al sondaggio')
plt.xlabel('Età')
plt.ylabel('Conteggio')
plt.show()

# Visualizza un conteggio delle risposte sulla soddisfazione
plt.figure(figsize=(8, 6))
sns.countplot(x='Soddisfazione', data=df, order=['Molto Soddisfatto', 'Soddisfatto', 'Insoddisfatto', 'Neutro'])
plt.title('Conteggio delle risposte sulla soddisfazione')
plt.xlabel('Soddisfazione')
plt.ylabel('Conteggio')
plt.xticks(rotation=45)
plt.show()
```

	Età	Soddisfazione	Numeric_Var
0	56	Molto Soddisfatto	41.651376
1	46	Molto Insoddisfatto	40.054054
2	32	Neutro	41.747368
3	60	Neutro	41.747368
4	25	Molto Insoddisfatto	40.054054



## Conteggio delle risposte sulla soddisfazione



```
In [38]: np.random.rand(100, 5)
```

```
Out[38]: array([[3.91482110e-01, 5.31857480e-01, 6.66191124e-02, 2.29025391e-01,
   5.42849311e-01],
 [4.31528143e-01, 3.32815786e-01, 7.30549137e-01, 6.93717607e-01,
  1.66730760e-01],
 [8.78629115e-01, 4.95423626e-01, 7.41460911e-01, 5.73150849e-01,
  9.97692616e-01],
 [7.52403120e-01, 7.06980461e-01, 7.78571939e-01, 1.43127986e-01,
  2.04544593e-01],
 [7.14064082e-01, 4.93981487e-01, 7.54635292e-01, 1.02916031e-01,
  5.36481351e-01],
 [3.78821566e-01, 4.57000219e-01, 6.03957872e-01, 5.02288347e-01,
  5.39860637e-01],
 [4.86357482e-01, 4.08955177e-01, 7.71881981e-01, 1.22030723e-02,
  5.98442701e-01],
 [5.65508347e-01, 7.16179075e-01, 5.99029365e-01, 8.26798828e-01,
  9.59074795e-01],
 [3.42524257e-01, 2.27350983e-01, 4.23596862e-01, 2.87929616e-01,
  6.14950263e-01],
 [9.11852409e-01, 1.39116194e-01, 1.00794603e-01, 2.56015532e-01,
  -0.00000000e+00],
```

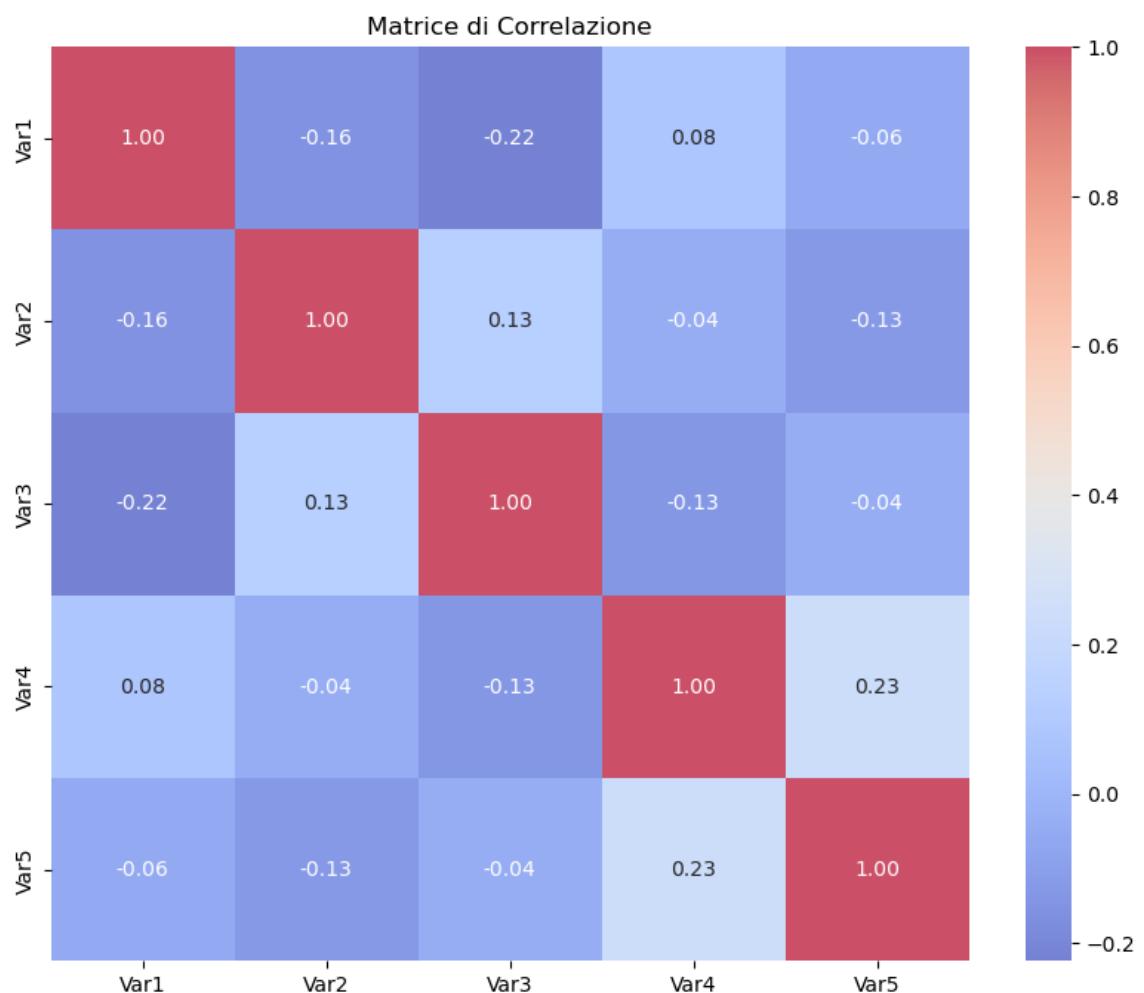
```
In [14]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Genera un dataset di esempio con variabili numeriche
np.random.seed(42)
data = pd.DataFrame(np.random.rand(100, 5), columns=['Var1', 'Var2', 'Var3'])

# Aggiungi alcune variabili categoriche generate casualmente
data['Categoria1'] = np.random.choice(['A', 'B', 'C'], size=100)
data['Categoria2'] = np.random.choice(['X', 'Y'], size=100)

# Calcola la matrice di correlazione tra tutte le variabili numeriche
correlation_matrix = data.corr()

# Visualizza la matrice di correlazione come heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", alpha=0.8)
plt.title("Matrice di Correlazione")
plt.show()
```



```
In [ ]:
```

```
In [ ]:
```

In [ ]:

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

# Impostare il seed per rendere i risultati riproducibili
np.random.seed(41)

# Creare un dataframe vuoto
df = pd.DataFrame()

# Generare dati casuali
n_rows = 10000
df['CatCol1'] = np.random.choice(['A', 'B', 'C'], size=n_rows)
df['CatCol2'] = np.random.choice(['X', 'Y'], size=n_rows)
df['NumCol1'] = np.random.randn(n_rows)
df['NumCol2'] = np.random.randint(1, 100, size=n_rows)
df['NumCol3'] = np.random.uniform(0, 1, size=n_rows)

# Calcolare il numero totale di missing values desiderati
total_missing_values = int(0.03 * n_rows * len(df.columns))

# Introdurre missing values casuali
for column in df.columns:
    num_missing_values = np.random.randint(0, total_missing_values + 1)
    missing_indices = np.random.choice(n_rows, size=num_missing_values, replace=False)
    df.loc[missing_indices, column] = np.nan
df
```

Out[2]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
1	A	Y	1.945879	28.0	0.936825
2	C	X	0.988834	42.0	0.751516
3	A	Y	-0.181978	73.0	0.950696
4	B	X	2.080615	74.0	0.903045
...	...	...	...	...	...
9995	C	Y	1.352114	61.0	0.728445
9996	C	Y	1.143642	67.0	0.605930
9997	A	X	-0.665794	54.0	0.071041
9998	C	Y	0.004278	NaN	NaN
9999	A	X	0.622473	95.0	0.751384

10000 rows × 5 columns

In [4]:

```
# Identificazione delle righe con dati mancanti
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
len(righe_con_dati_mancanti)
```

Out[4]: 3648

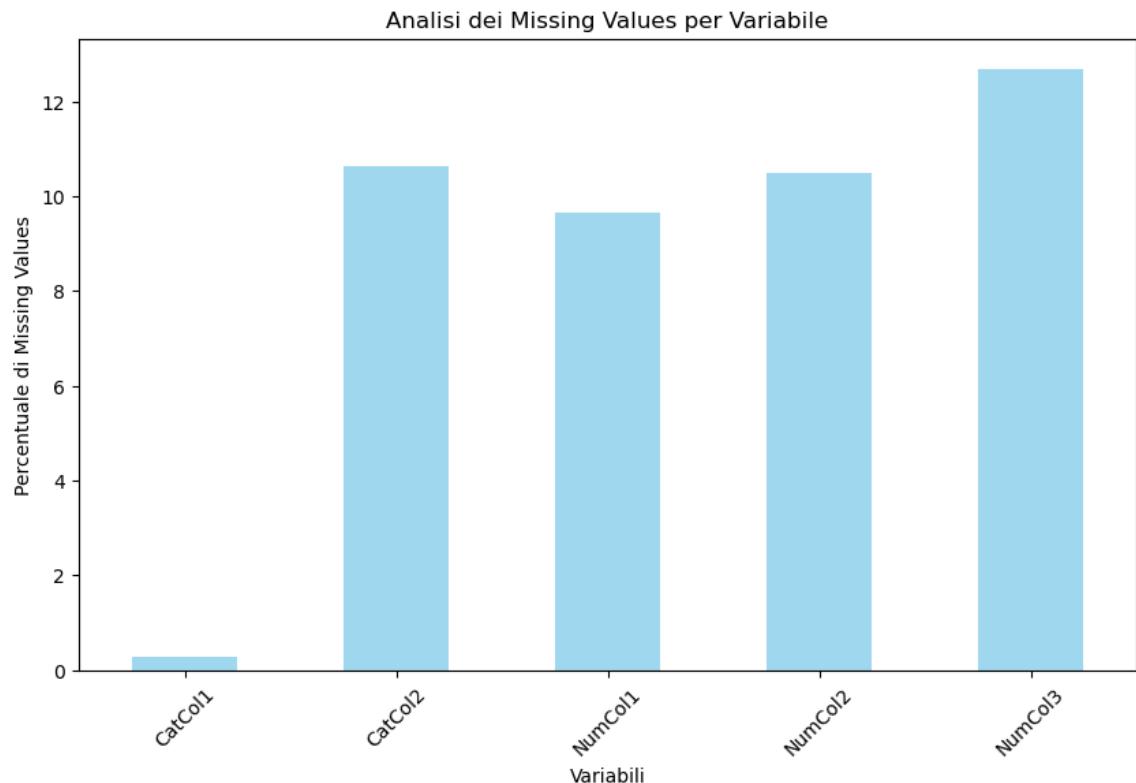
In [ ]:

```
In [9]: missing_percent = (df.isnull().sum() / len(df)) * 100  
missing_percent
```

```
Out[9]: CatCol1      0.29  
CatCol2      10.63  
NumCol1      9.67  
NumCol2      10.48  
NumCol3      12.69  
dtype: float64
```

```
In [11]: import matplotlib.pyplot as plt
```

```
# Crea il grafico a barre  
plt.figure(figsize=(10, 6))  
missing_percent.plot(kind='bar', color='skyblue', alpha=0.8)  
plt.xlabel('Variabili')  
plt.ylabel('Percentuale di Missing Values')  
plt.title('Analisi dei Missing Values per Variabile')  
plt.xticks(rotation=45)  
plt.show()
```

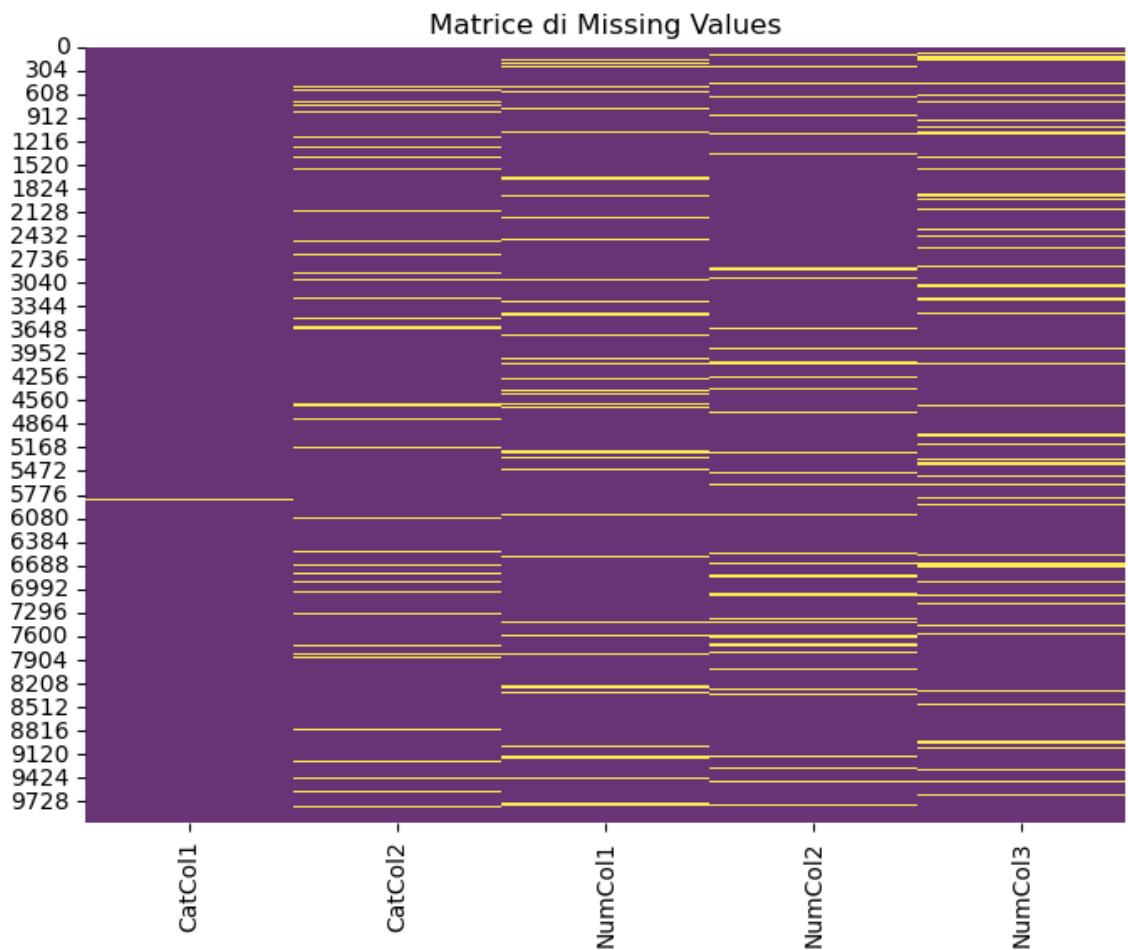


```
In [13]: import seaborn as sns

missing_matrix = df.isnull()

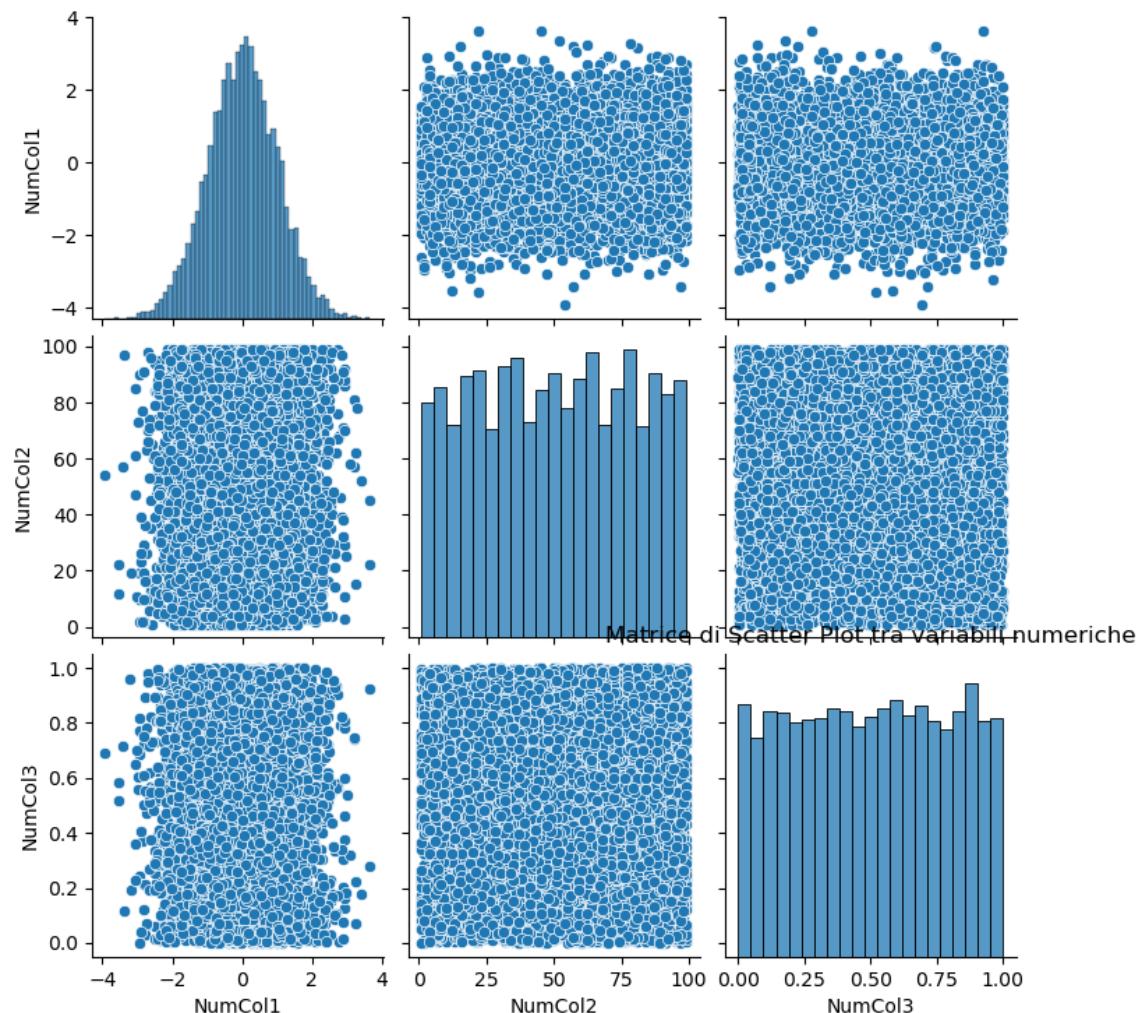
# Crea una heatmap colorata
plt.figure(figsize=(8, 6))
sns.heatmap(missing_matrix, cmap='viridis', cbar=False, alpha=0.8)
plt.title('Matrice di Missing Values')
plt.xticks(rotation=90)

plt.show()
```



```
In [ ]:
```

```
In [14]: # Visualizza la distribuzione delle variabili numeriche  
numeric_features = df.select_dtypes(include=[np.number])  
# Visualizza una matrice di scatter plot tra le variabili numeriche  
sns.pairplot(df[numeric_features.columns])  
plt.title('Matrice di Scatter Plot tra variabili numeriche')  
plt.show()
```



In [5]: # Elimina le righe in cui entrambe le features categoriche hanno valori NaN  
`df = df.dropna(subset=["CatCol1", "CatCol2"], how='all')`  
`df`

Out[5]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
1	A	Y	1.945879	28.0	0.936825
2	C	X	0.988834	42.0	0.751516
3	A	Y	-0.181978	73.0	0.950696
4	B	X	2.080615	74.0	0.903045
...	...	...	...	...	...
9995	C	Y	1.352114	61.0	0.728445
9996	C	Y	1.143642	67.0	0.605930
9997	A	X	-0.665794	54.0	0.071041
9998	C	Y	0.004278	NaN	NaN
9999	A	X	0.622473	95.0	0.751384

9995 rows × 5 columns

In [47]: `df = df.dropna(subset=["NumCol1", "NumCol2", "NumCol3"], how='all')`  
`df`

Out[47]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
1	A	Y	1.945879	28.0	0.936825
2	C	X	0.988834	42.0	0.751516
3	A	Y	-0.181978	73.0	0.950696
4	B	X	2.080615	74.0	0.903045
...	...	...	...	...	...
9995	C	Y	1.352114	61.0	0.728445
9996	C	Y	1.143642	67.0	0.605930
9997	A	X	-0.665794	54.0	0.071041
9998	C	Y	0.004278	NaN	NaN
9999	A	X	0.622473	95.0	0.751384

9975 rows × 5 columns

In [ ]:

```
In [48]: numeric_cols = df.select_dtypes(include=['number'])
categorical_cols = df.select_dtypes(exclude=['number'])

# Sostituisci i missing values nelle colonne categoriche con la moda utilizzando .mode()
df.loc[:, categorical_cols.columns] = df[categorical_cols.columns].fillna(df.mode().iloc[0])

# Calcola la media condizionata solo per le colonne numeriche con dati mancanti
conditional_means = df[numeric_cols.columns].fillna(df.groupby('CatCol1')[numeric_cols].mean())

# Aggiorna le colonne numeriche con la media condizionata utilizzando .loc
df.loc[:, numeric_cols.columns] = conditional_means

# Stampa il DataFrame risultante
print(df)
```

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0		A	Y 0.440877	49.000000	0.246007
1		A	Y 1.945879	28.000000	0.936825
2		C	X 0.988834	42.000000	0.751516
3		A	Y -0.181978	73.000000	0.950696
4		B	X 2.080615	74.000000	0.903045
...	...	...	...	...	...
9995		C	Y 1.352114	61.000000	0.728445
9996		C	Y 1.143642	67.000000	0.605930
9997		A	X -0.665794	54.000000	0.071041
9998		C	Y 0.004278	49.845018	0.489352
9999		A	X 0.622473	95.000000	0.751384

[9975 rows x 5 columns]

C:\Users\utente\AppData\Local\Temp\ipykernel\_13024\1333295202.py:5: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) ([https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy))

C:\Users\utente\AppData\Local\Temp\ipykernel\_13024\1333295202.py:11: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) ([https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy))

In [ ]:

```
In [49]: import pandas as pd
import numpy as np

# Impostare il seed per rendere i risultati riproducibili
np.random.seed(41)

# Creare un dataframe vuoto
df = pd.DataFrame()

# Generare dati casuali
n_rows = 10000000
df['CatCol1'] = np.random.choice(['A', 'B', 'C'], size=n_rows)
df['CatCol2'] = np.random.choice(['X', 'Y'], size=n_rows)
df['NumCol1'] = np.random.randn(n_rows)
df['NumCol2'] = np.random.randint(1, 100, size=n_rows)
df['NumCol3'] = np.random.uniform(0, 1, size=n_rows)

# Calcolare il numero totale di missing values desiderati
total_missing_values = int(0.05 * n_rows * len(df.columns))

# Introdurre missing values casuali
for column in df.columns:
    num_missing_values = np.random.randint(0, total_missing_values + 1)
    missing_indices = np.random.choice(n_rows, size=num_missing_values, replace=False)
    df.loc[missing_indices, column] = np.nan

# Elimina le righe in cui entrambe le features categoriche hanno valori NaN
df = df.dropna(subset=["CatCol1", "CatCol2"], how='all')
df = df.dropna(subset=["NumCol1", "NumCol2", "NumCol3"], how='all')

numeric_cols = df.select_dtypes(include=['number'])
categorical_cols = df.select_dtypes(exclude=['number'])

# Sostituisci i missing values nelle colonne categoriche con la moda utilizzando .mode()
df.loc[:, categorical_cols.columns] = df[categorical_cols.columns].fillna(df.mode().iloc[0])

# Calcola la media condizionata solo per le colonne numeriche con dati mancanti
conditional_means = df[numeric_cols.columns].fillna(df.groupby('CatCol1')[numeric_cols.columns].mean())

# Aggiorna le colonne numeriche con la media condizionata utilizzando .loc
df.loc[:, numeric_cols.columns] = conditional_means

# Stampa il DataFrame risultante
print(df)
```

```
CatCol1  CatCol2  NumCol1  NumCol2  NumCol3
0          A        Y -0.391604   98.0  0.409815
1          A        X  0.000551   19.0  0.886592
2          C        Y  1.266001   52.0  0.848556
3          A        X  0.449617   70.0  0.546525
4          B        X  0.742505   72.0  0.467257
...
9999995    A        Y  0.464663    7.0  0.992815
9999996    A        X  0.149775   13.0  0.731368
9999997    C        Y -0.608376    1.0  0.606349
9999998    C        Y  0.000101   69.0  0.115812
9999999    B        Y  1.666715   76.0  0.245699
```

[9635330 rows x 5 columns]

In [ ]:



```
In [1]: pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\xdmat\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\xdmat\anaconda3\lib\site-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\xdmat\anaconda3\lib\site-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\xdmat\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\xdmat\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

```
In [6]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Creare dati casuali per altezze (variabile indipendente) e pesi (variabile dipendente)
np.random.seed(0)
altezze = np.random.normal(160, 10, 100)
pesi = 0.5 * altezze + np.random.normal(0, 5, 100)

# Suddividere il dataset in training set (70%) e test set (30%)
X_train, X_test, y_train, y_test = train_test_split(altezze, pesi, test_size=0.3)

# Stampare le dimensioni dei training set e test set
print("Dimensioni del Training Set (altezze e pesi):", X_train.shape, y_train.shape)
print("Dimensioni del Test Set (altezze e pesi):", X_test.shape, y_test.shape)
```

Dimensioni del Training Set (altezze e pesi): (70,) (70,)
Dimensioni del Test Set (altezze e pesi): (30,) (30,)

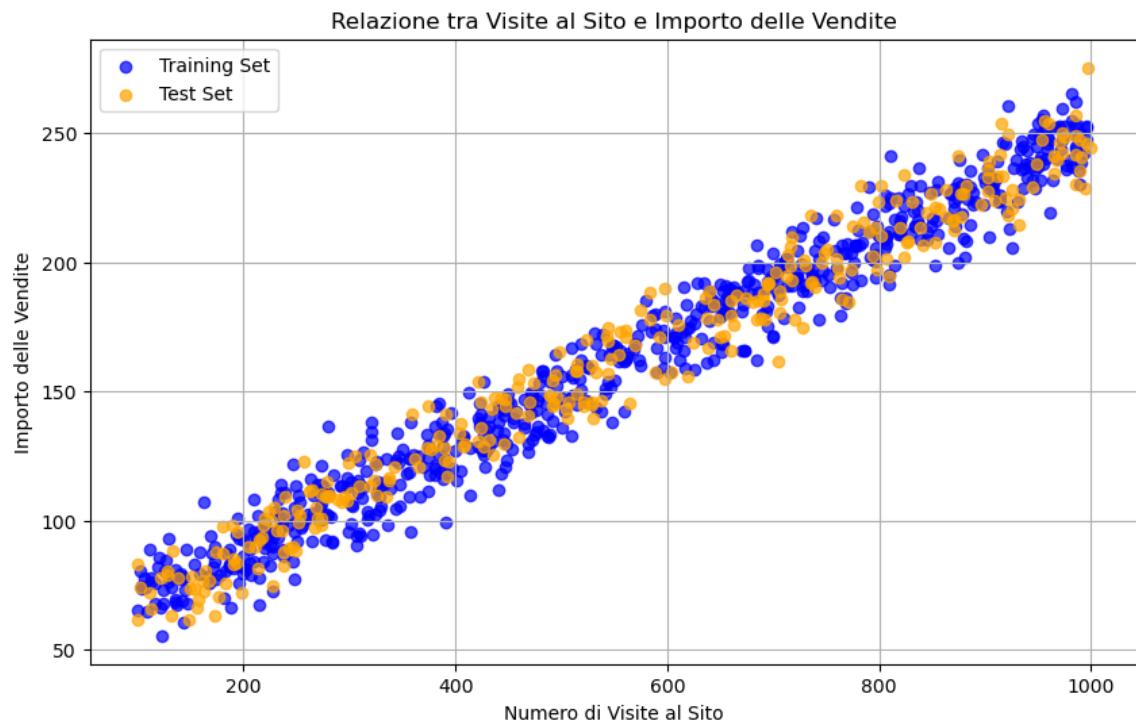
```
In [7]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Creazione di dati casuali per visite al sito web e importo delle vendite
np.random.seed(0)
visite_al_sito = np.random.randint(100, 1000, 1000)
importo_vendite = 50 + 0.2 * visite_al_sito + np.random.normal(0, 10, 1000)

# Suddivisione del dataset in training set (70%) e test set (30%)
X_train, X_test, y_train, y_test = train_test_split(visite_al_sito, importo_vendite, test_size=0.3, random_state=0)

# Creazione di un grafico a dispersione
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, label='Training Set', color='blue', alpha=0.7)
plt.scatter(X_test, y_test, label='Test Set', color='orange', alpha=0.7)
plt.xlabel('Numero di Visite al Sito')
plt.ylabel('Importo delle Vendite')
plt.title('Relazione tra Visite al Sito e Importo delle Vendite')
plt.legend()
plt.grid(True)
plt.show()

# Stampare le dimensioni dei training set e test set
print("Dimensioni del Training Set (visite al sito e importo delle vendite):", X_train.shape)
print("Dimensioni del Test Set (visite al sito e importo delle vendite):", X_test.shape)
```



Dimensioni del Training Set (visite al sito e importo delle vendite): (700,)  
 Dimensioni del Test Set (visite al sito e importo delle vendite): (300,)

```
In [9]: from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(1)
# Supponiamo di avere un dataset con feature X e target y
X = np.random.rand(100, 2) # Dati del dataset (100 campioni, 2 feature)
y = np.random.choice(['A', 'B'], size=100) # Etichette di classe casuali
# Calcola le proporzioni delle classi nel dataset originale
proporzione_classe_A = sum(y == 'A') / len(y)
proporzione_classe_B = 1 - proporzione_classe_A
# Eseguire uno split stratificato con una proporzione specificata
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# Calcola le proporzioni delle classi nel training set e nel test set
proporzione_classe_A_train = sum(y_train == 'A') / len(y_train)
proporzione_classe_B_train = 1 - proporzione_classe_A_train

proporzione_classe_A_test = sum(y_test == 'A') / len(y_test)
proporzione_classe_B_test = 1 - proporzione_classe_A_test

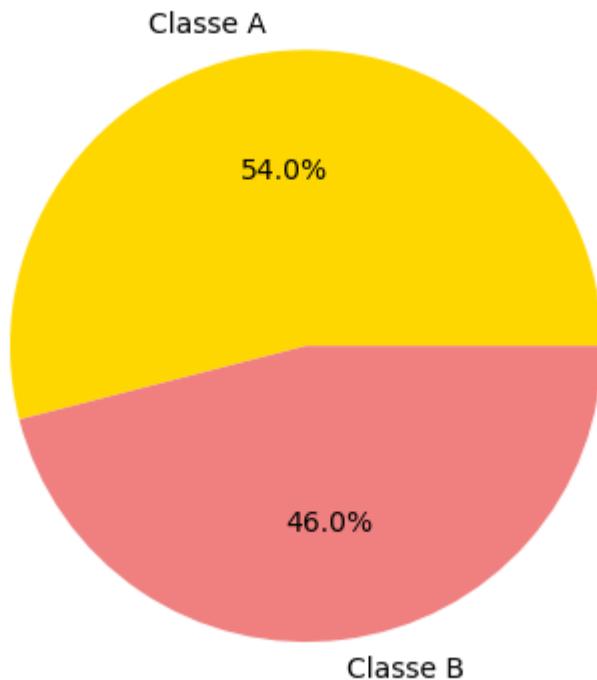
# Stampa delle proporzioni
print("Proporzione Classe A nel data Set completo:", proporzione_classe_A)
print("Proporzione Classe B nel data Set completo:", proporzione_classe_B)
print("Proporzione Classe A nel Training Set:", proporzione_classe_A_train)
print("Proporzione Classe B nel Training Set:", proporzione_classe_B_train)
print("Proporzione Classe A nel Test Set:", proporzione_classe_A_test)
print("Proporzione Classe B nel Test Set:", proporzione_classe_B_test)
```

```
Proporzione Classe A nel data Set completo: 0.54
Proporzione Classe B nel data Set completo: 0.45999999999999996
Proporzione Classe A nel Training Set: 0.5285714285714286
Proporzione Classe B nel Training Set: 0.4714285714285714
Proporzione Classe A nel Test Set: 0.5666666666666667
Proporzione Classe B nel Test Set: 0.4333333333333335
```

In [10]:

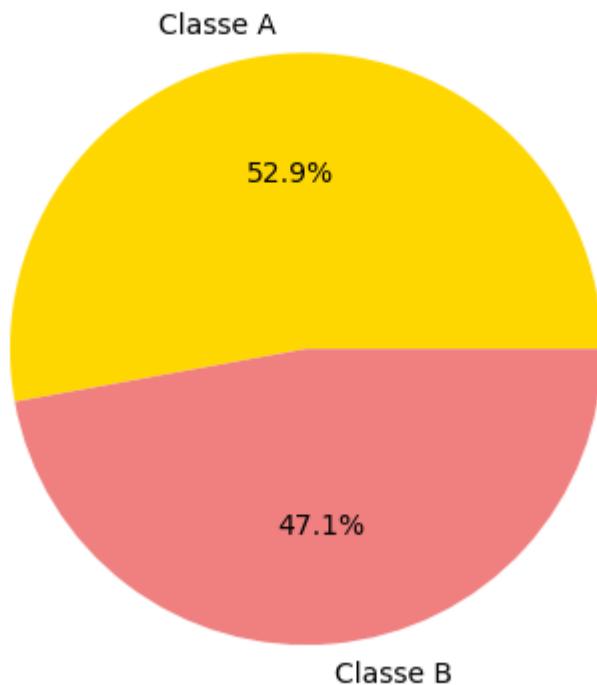
```
# Etichette delle classi
labels = ['Classe A', 'Classe B']
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']
# Crea un grafico a torta con etichette
plt.pie([proporzione_classe_A, proporzione_classe_B], labels=labels, colors=colors)
plt.title('Proporzione delle Classi nel Set')
plt.show()
```

Proporzione delle Classi nel Set



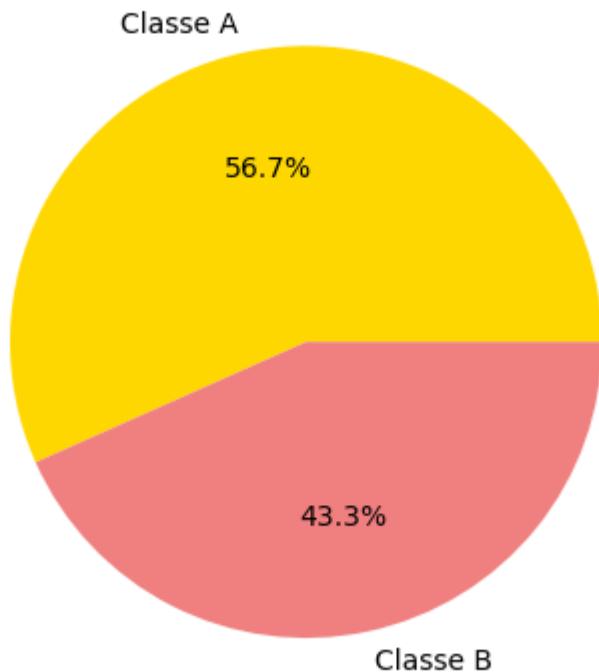
```
In [13]: # Etichette delle classi
labels = ['Classe A', 'Classe B']
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']
# Crea un grafico a torta con etichette
plt.pie([proporzione_classe_A_train, proporzione_classe_B_train], labels=labels, colors=colors)
plt.title('Proporzione delle Classi nel Train')
plt.show()
```

Proporzione delle Classi nel Train



```
In [14]: # Etichette delle classi
labels = ['Classe A', 'Classe B']
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']
# Crea un grafico a torta con etichette
plt.pie([proporzione_classe_A_test, proporzione_classe_B_test], labels=labels)
plt.title('Proporzione delle Classi nel Test')
plt.show()
```

Proporzione delle Classi nel Test



```
In [15]: import random
import numpy as np

dataset=[]
# Creazione di un dataset di 1000 elementi (ad esempio, dati casuali)
popolazione =24000000
for i in range(popolazione):
    dataset.append(random.randint(0, 100000))

campione = int(round(0.3 * popolazione))# Estrazione di un campione casuale
campione_casuale = random.sample(dataset, campione)

# Calcolo della media e della deviazione standard del campione
media_campione = np.mean(campione_casuale)
deviazione_standard_campione = np.std(campione_casuale)

# Calcolo della media e della deviazione standard del dataset completo
media_dataset = np.mean(dataset)
deviazione_standard_dataset = np.std(dataset)

print(f"Media del campione casuale: {media_campione: .2f}")
print(f"Deviazione standard del campione casuale: {deviazione_standard_campione: .2f}")
print(f"Media del dataset completo: {media_dataset: .2f}")
print(f"Deviazione standard del dataset completo: {deviazione_standard_dataset: .2f}")

Media del campione casuale: 49999.58
Deviazione standard del campione casuale: 28864.89
Media del dataset completo: 50000.40
Deviazione standard del dataset completo: 28867.95
```

```
In [ ]:
```