RIASSUNTO MODULO 1

February 9, 2024

1 ESERCITAZIONE 1

1.1 Primi Codici

1.1.1 Saluto

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

```
Ciao, mondo!
```

1.1.2 Stampa Nome

[3]:
```
nome="Mattia"
print(nome)
```

```
Mattia
```

1.1.3 Variabili e Input Utente

[2]:
```
nome = input("Inserisci il tuo nome: ")
print("Ciao,", nome,"!")
```

```
Inserisci il tuo nome: Mattia
Ciao, Mattia !
```

1.1.4 Input della Via

[6]:
```
via=input("inserisci nome della via ")
print("hai inserito",via)
```

```
inserisci nome della via Po
hai inserito Po
```

1.1.5 Input Ripetuto

[7]:
```
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: Mattia
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
Ciao, Mattia !
```

1.2 Calcolatrice Python

1.2.1 Addizione

[1]:
```python
# Richiesta all'utente di inserire il primo numero intero
numero1 = int(input("Inserisci il primo numero: "))

# Richiesta all'utente di inserire il secondo numero intero
numero2 = int(input("Inserisci il secondo numero: "))

# Calcolo della somma dei due numeri inseriti
somma = numero1 + numero2

# Stampa del risultato della somma
print("La somma è:", somma)
```

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

1.2.2 Sottrazione

[2]:
```python
# Calcolo della sottrazione tra numero1 e numero2
sottrazione = numero1 - numero2

# Stampa del risultato della sottrazione
print("La sottrazione è:", sottrazione)
```

```
La sottrazione è: -1
```

1.2.3 Moltiplicazione

[2]:
```python
# Richiesta all'utente di inserire il primo numero intero
numero1 = int(input("Inserisci il primo numero: "))

# Richiesta all'utente di inserire il secondo numero intero
numero2 = int(input("Inserisci il secondo numero: "))
```

```python
# Calcolo della moltiplicazione tra numero1 e numero2
moltiplicazione = numero1 * numero2

# Stampa del risultato della moltiplicazione
print("La moltiplicazione è:", moltiplicazione)
```

```
Inserisci il primo numero: 4
Inserisci il secondo numero: 5
La moltiplicazione è: 20
```

1.2.4 Divisione

[3]:
```python
# Calcolo della divisione tra numero1 e numero2
divisione = numero1 / numero2

# Stampa del risultato della divisione
print("La divisione è:", divisione)
```

```
La divisione è: 0.8
```

1.3 Loop e Ripetizione

1.3.1 Stampa Numeri da 1 a 10

[4]:
```python
# Iterazione attraverso i numeri da 1 a 10 (incluso)
for numero in range(1, 10+1):
    # Stampa del numero corrente
    print(numero)
```

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

1.3.2 Calcolatrice Python con Decisioni

[6]:
```python
# Richiesta all'utente di inserire l'operazione desiderata
operazione = input("Inserisci l'operazione (+, -, *, /): ")

# Richiesta all'utente di inserire il primo numero (come numero decimale)
numero1 = float(input("Inserisci il primo numero: "))
```

```python
# Richiesta all'utente di inserire il secondo numero (come numero decimale)
numero2 = float(input("Inserisci il secondo numero: "))

# Utilizzo di condizioni per eseguire l'operazione corrispondente
if operazione == "+":
    risultato = numero1 + numero2
elif operazione == "-":
    risultato = numero1 - numero2
elif operazione == "*":
    risultato = numero1 * numero2
elif operazione == "/":
    # Attenzione alla divisione per zero
    if numero2 != 0:
        risultato = numero1 / numero2
    else:
        risultato = "Errore: Divisione per zero"
else:
    risultato = "Operazione non valida"

# Stampa del risultato ottenuto
print("Il risultato è:", risultato)
```

```
Inserisci l'operazione (+, -, *, /): /
Inserisci il primo numero: 56
Inserisci il secondo numero: 65
Il risultato è: 0.8615384615384616
```

1.3.3 Contare fino a N

[7]:
```python
# Richiesta all'utente di inserire un numero intero positivo
n = int(input("Inserisci un numero intero positivo:"))

# Iterazione attraverso i numeri da 5 a n (incluso)
for numero in range(5, n + 1):
    # Stampa del numero corrente
    print(numero)
```

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

15
16
17
18

1.3.4 Calcolare la Somma

[8]:
```
# Richiesta all'utente di inserire un numero intero positivo
n = int(input("Inserisci un numero intero positivo: "))

# Inizializzazione della variabile somma
somma = 0

# Iterazione attraverso i numeri da 1 a n (incluso)
for numero in range(1, n + 1):
    # Aggiornamento della somma con il numero corrente
    somma += numero

# Stampa del risultato della somma
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
```

1.3.5 Calcolare il Quadrato dei Primi Numeri

[9]:
```
# Richiesta all'utente di inserire un numero intero positivo
n = int(input("Inserisci un numero intero positivo: "))

# Stampa del messaggio introduttivo
print("Quadrati dei primi", n, "numeri:")

# Iterazione attraverso i numeri da 1 a n (incluso)
for numero in range(1, n + 1):
    # Calcolo del quadrato del numero corrente
    quadrato = numero ** 2

    # Stampa del risultato
    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
```

1.3.6 Verificare la Parità

[11]:
```
# Richiesta all'utente di inserire un numero
numero = int(input("Inserisci un numero: "))

# Verifica se il numero è pari o dispari
if numero % 2 == 0:
    print(numero, "è un numero pari.")
else:
    print(numero, "è un numero dispari.")
```

```
Inserisci un numero: 4
4 è un numero pari.
```

1.3.7 Calcolare il Fattoriale

[12]:
```
# Richiesta all'utente di inserire un numero intero positivo
n = int(input("Inserisci un numero intero positivo: "))

# Inizializzazione della variabile fattoriale
fattoriale = 1

# Calcolo del fattoriale di n
for numero in range(1, n + 1):
    fattoriale *= numero

# Stampa del risultato del fattoriale
print("Il fattoriale di", n, "è:", fattoriale)
```

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

1.3.8 Calcolare la Media di una Lista di Numeri

[13]:
```
# Inizializza una lista vuota per contenere i numeri
numeri = []

# Chiede all'utente quanti numeri vuole inserire
n = int(input("Quanti numeri vuoi inserire? "))

# Ciclo per ottenere i numeri dall'utente e aggiungerli alla lista
for i in range(n):
    numero = float(input("Inserisci un numero: "))  # float = floating point␣
 ↪punti e virgola movibili
    numeri.append(numero)

# Calcola la media dei numeri nella lista
media = sum(numeri) / len(numeri)  # len = lunghezza, sum = somma
```

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

```
Quanti numeri vuoi inserire? 5
Inserisci un numero: 7
Inserisci un numero: 8
Inserisci un numero: 9
Inserisci un numero: 6
Inserisci un numero: 5
La media dei numeri inseriti è: 7.0 la lista completa è: [7.0, 8.0, 9.0, 6.0,
5.0]
```

1.3.9 Gioco dell'Indovinello

[15]:
```
# Importa il modulo random per generare numeri casuali
import random

# Genera un numero casuale da indovinare compreso tra 1 e 100
numero_da_indovinare = random.randint(1, 100)  # randint = numero random␣
 ↪compreso tra i due estremi
tentativi = 0

# Ciclo while per continuare a chiedere all'utente di indovinare
while True:
    # Richiede all'utente di fare un tentativo
    tentativo = int(input("Indovina il numero (1-100): "))
    tentativi += 1

    # Verifica se il tentativo è corretto, troppo grande o troppo piccolo
    if tentativo == numero_da_indovinare:
        print("Bravo! Hai indovinato il numero", numero_da_indovinare, "in",␣
 ↪tentativi, "tentativi.")
        break  # Interrompe il ciclo while se il numero è stato indovinato
    elif tentativo < numero_da_indovinare:
        print("Il numero è più grande.")
    else:
        print("Il numero è più piccolo.")
```

```
Indovina il numero (1-100): 44
Il numero è più grande.
Indovina il numero (1-100): 66
Il numero è più grande.
Indovina il numero (1-100): 99
Il numero è più piccolo.
Indovina il numero (1-100): 77
Il numero è più grande.
Indovina il numero (1-100): 88
Il numero è più piccolo.
```

```
Indovina il numero (1-100): 80
Il numero è più grande.
Indovina il numero (1-100): 83
Bravo! Hai indovinato il numero 83 in 7 tentativi.
```

1.3.10 Gioco del Morra Cinese

[18]:
```python
# Importa il modulo random per la scelta casuale del computer
import random

# Definisce le possibili mosse nel gioco
mosse = ["carta", "forbici", "sasso"]

# Il computer fa una scelta casuale tra le mosse possibili
computer_mossa = random.choice(mosse)

# Stampa un messaggio di benvenuto
print("Benvenuti al Gioco della Morra Cinese!")

# Chiede all'utente di scegliere una mossa
scelta_giocatore = input("Scegli la tua mossa (carta, forbici, sasso): ")

# Verifica se la mossa dell'utente è permessa
if scelta_giocatore not in mosse:
    print("Mossa non permessa")
else:
    # Stampa la mossa del computer
    print("Il computer ha scelto:", computer_mossa)

    # Confronta le mosse dell'utente e del computer per determinare il risultato
    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 della Morra Cinese!
Scegli la tua mossa (carta, forbici, sasso): carta
Il computer ha scelto: sasso
Hai vinto
```

1.3.11 Calcolo del Fattoriale

[19]:
```python
# Richiesta all'utente di inserire un numero intero
n = int(input("Inserisci un numero intero: "))

# Inizializzazione della variabile fattoriale
fattoriale = 1

# Verifica e calcolo del fattoriale
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

# Stampa del risultato del fattoriale
print(f"Il fattoriale di {n} è {fattoriale}")
```

```
Inserisci un numero intero: 4
Il fattoriale di 4 è 24
```

1.3.12 Calcolo del Fattoriale con Gestione dei Numeri Negativi

[21]:
```python
# Chiede all'utente di inserire un numero intero
n = int(input("Inserisci un numero intero: "))

# Inizializza la variabile per il calcolo del fattoriale
fattoriale = 1

# Verifica se il numero è negativo
if n < 0:
    print("Il numero è negativo.")
# Se il numero è zero, il fattoriale è per convenzione 1
elif n == 0:
    print("Il fattoriale di zero è 1.")
else:
    # Calcola il fattoriale usando un ciclo for
    for numero in range(1, n + 1):
        fattoriale *= numero

# Stampa il risultato del fattoriale
print(f"Il fattoriale di {n} è {fattoriale}")
```

```
Inserisci un numero intero: -5
Il numero è negativo.
Il fattoriale di -5 è 1
```

2 ESERCITAZIONE 2

2.1 Strumenti Matematici: Generazione di Numeri e Calcoli

2.1.1 Calcolo della Somma dei Numeri Pari fino a "N"

[22]:
```
# 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
```

2.1.2 Generazione di una Lista dei Numeri Pari fino a "N"

[25]:
```
#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: 9
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

2.1.3 Conteggio delle Voci in una Frase o una Parola

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

# 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.
```

2.1.4 Gioco dell'Indovinello

[38]:
```
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␣
 ↪alla prossima!")
```

```
Indovina il numero del dado (da 1 a 6): 5
Mi dispiace, il numero del dado era 6. Meglio fortuna alla prossima!
```

2.1.5 Simulatore di Crescita della Popolazione

[39]:
```
# Inizializzazione della popolazione e degli anni
popolazione = int(input("Inserisci 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 di natalità: "))
tasso_mortalita = float(input("Inserisci tasso di mortalità: "))

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

```
    # Stampa della popolazione simulata per ogni anno
    print(f"Anno {anno+1}: Popolazione = {int(popolazione)}")

print("Simulazione completata.")
```

```
Inserisci popolazione iniziale: 41000000
Inserisci numero di anni da simulare: 4
Inserisci tasso di natalità: 5
Inserisci tasso di mortalità: 7
Anno 1: Popolazione = 40180000
Anno 2: Popolazione = 39376400
Anno 3: Popolazione = 38588872
Anno 4: Popolazione = 37817094
Simulazione completata.
```

2.1.6 Risolutore di Equazioni di Secondo Grado

[40]:
```
# importazione del modulo math
import math

# Messaggio di benvenuto
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': 6
Inserisci il coefficiente 'b': 8
Inserisci il coefficiente 'c': 9
L'equazione ha due soluzioni complesse: x1 = -0.67 + 1.03i e x2 = -0.67 - 1.03i
```

2.1.7 Stampa Data e Ora Attuali

[41]:
```python
# Importazione del modulo datetime
import datetime

# Ottenimento della data e ora attuali
today = datetime.datetime.today()

# Stampa della data e ora attuali nel formato specificato
print(f"Oggi è il giorno: {today:%d %m %Y} ore: {today:%H %M %S}")
```

```
Oggi è il giorno: 26 01 2024 ore: 12 52 24
```

2.1.8 Convertitore Universale di Unità di Misura

[1]:
```python
# Messaggio di benvenuto
print("Benvenuto nel Convertitore di Unità di Misura!")

# Richiesta all'utente di cosa desidera convertire
scelta = input("Cosa desideri convertire? (metri/piedi/chilogrammi/libbre): ").
 ↪lower()

# Logica di conversione basata sulla scelta dell'utente
if scelta == "metri":
    valore = float(input("Inserisci il valore in metri: "))
    risultato = valore * 3.28084
    print(f"{valore} metri corrispondono a {risultato:.2f} piedi.")

elif scelta == "piedi":
    valore = float(input("Inserisci il valore in piedi: "))
    risultato = valore / 3.28084
    print(f"{valore} piedi corrispondono a {risultato:.2f} metri.")

elif scelta == "chilogrammi":
    valore = float(input("Inserisci il valore in chilogrammi: "))
    risultato = valore * 2.20462
    print(f"{valore} chilogrammi corrispondono a {risultato:.2f} libbre.")

elif scelta == "libbre":
    valore = float(input("Inserisci il valore in libbre: "))
    risultato = valore / 2.20462
```

```
    print(f"{valore} libbre corrispondono a {risultato:.2f} chilogrammi.")

else:
    print("Scelta non valida. Scegli tra 'metri', 'piedi', 'chilogrammi' o␣
 ↪'libbre'.")
```

```
Benvenuto nel Convertitore di Unità di Misura!
Cosa desideri convertire? (metri/piedi/chilogrammi/libbre): metri
Inserisci il valore in metri: 54
54.0 metri corrispondono a 177.17 piedi.
```

2.1.9 Calcolo dell'n-esimo Numero di Fibonacci

[2]:
```
# 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, c = b, c, a + b
    risultato = c

# Stampare l'n-esimo numero di Fibonacci
print(f"L'n-esimo numero di Fibonacci è: {risultato}")
```

```
Inserisci un numero n per calcolare l'n-esimo numero di Fibonacci: 45
L'n-esimo numero di Fibonacci è: 128801
```

2.2 FUNZIONI CUSTOM

2.2.1 Generatore di Serie di Fibonacci

[2]:
```
def Fibonacci(n):
    # Inizializza la serie di Fibonacci con i primi due termini
    fib_series = [0, 1]

    # Continua ad aggiungere nuovi termini finché la lunghezza della serie è␣
 ↪inferiore a n
    while len(fib_series) < n:
```

```
        # Aggiunge il nuovo termine sommando gli ultimi due termini della serie
        fib_series.append(fib_series[-1] + fib_series[-2])

    # Restituisce la serie di Fibonacci fino all'n-esimo termine
    return fib_series
```

[14]:
```
fibonacci(15)
#In questo modo, la funzione restituirà una lista contenente i primi 15 termini␣
 ↪della serie di Fibonacci, e verranno stampati a schermo.
```

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

2.2.2 Generatore Interattivo di Serie di Fibonacci

[4]:
```
# Richiesta all'utente di inserire il numero di termini della serie di Fibonacci
n = int(input("Inserisci il numero di termini della serie di Fibonacci da␣
 ↪generare: "))

# Verifica se il numero inserito è positivo
if n <= 0:
    print("Inserisci un numero positivo.")
else:
    # Chiamata alla funzione Fibonacci per generare la serie
    result = Fibonacci(n)

    # Stampa la serie di Fibonacci generata
    print(result)
```

```
Inserisci il numero di termini della serie di Fibonacci da generare: 20
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181]
```

2.2.3 Calcolatore di Aree Geometriche

[5]:
```
import math

# Funzione per calcolare l'area di un cerchio
def calcola_area_cerchio(raggio):
    return math.pi * (raggio ** 2)

# Funzione per calcolare l'area di un rettangolo
def calcola_area_rettangolo(base, altezza):
    return base * altezza

# Funzione per calcolare l'area di un triangolo
def calcola_area_triangolo(base, altezza):
    return (base * altezza) / 2
```

[6]:
```
calcola_area_cerchio(10)
# Questa chiamata alla funzione calcola l'area del cerchio con un raggio di 10 e␣
 ↪stampa il risultato a schermo.
```

[6]:
```
314.1592653589793
```

Calcolatrice di Aree Geometriche

[7]:
```
import math

# Funzione per calcolare l'area di un cerchio
def calcola_area_cerchio(raggio):
    return math.pi * (raggio ** 2)

# Funzione per calcolare l'area di un rettangolo
def calcola_area_rettangolo(base, altezza):
    return base * altezza

# Funzione per calcolare l'area di un triangolo
def calcola_area_triangolo(base, altezza):
    return (base * altezza) / 2

# Messaggio di benvenuto
print("Benvenuto nella Calcolatrice di Aree!")

# Richiesta all'utente di cosa vuole calcolare
scelta = input("Vuoi calcolare l'area di un cerchio (c), rettangolo (r) o␣
 ↪triangolo (t)? ").lower()

# Logica di calcolo dell'area in base alla scelta dell'utente
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: 34
L'area del cerchio è 3631.68
```

2.2.4 Calcolatore di Interessi Composti

[8]:
```
# Funzione per calcolare gli interessi composti
def calcola_interessi(importo_iniziale, tasso_interesse, periodi_investimento):
    importo_finale = importo_iniziale * (1 + tasso_interesse / 100) **␣
 ↪periodi_investimento
    return importo_finale
```

2.2.5 Calcolatore di Interessi Composti

[2]:
```
# Funzione per calcolare gli interessi composti
def calcola_interessi(importo_iniziale, tasso_interesse, periodi_investimento):
    importo_finale = importo_iniziale * (1 + tasso_interesse / 100) **␣
 ↪periodi_investimento
    return importo_finale

# Messaggio di benvenuto
print("Benvenuto nel Calcolatore di Interessi!")

# Richiesta all'utente di inserire i dati dell'investimento
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): "))

# Calcolo dell'importo finale utilizzando la funzione calcola_interessi
importo_finale = calcola_interessi(importo, tasso, periodo)

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

```
Benvenuto nel Calcolatore di Interessi!
Inserisci l'importo iniziale: 12348
Inserisci il tasso di interesse annuale (%): 19
Inserisci il periodo di investimento (anni): 2
L'importo finale dopo 2 anni è di 17486.00 euro.
```

[6]:
```
# Chiamata alla funzione calcola_interessi con parametri specifici
calcola_interessi(10000000, 4, 10)
```

[6]:
```
14802442.849183444
```

2.2.6 Calcolatore di Forza Gravitazionale

[8]:
```python
# Funzione per calcolare la forza gravitazionale
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
```

[10]:
```python
# Esempio di utilizzo
massa_terra = 5.972e24  # kg
massa_luna = 7.342e22  # kg
distanza_terra_luna = 384400000  # metri

# Calcolo della forza gravitazionale
forza = forza_gravitazionale(massa_terra, massa_luna, distanza_terra_luna)

# Stampa del risultato
print(f"Forza gravitazionale tra la Terra e la Luna: {forza} Newton")
```

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

2.2.7 Risolutore di Anagrammi

[2]:
```python
# Stampa un messaggio di benvenuto
print("Benvenuto nel Risolutore di Anagrammi!")

# Funzione per trovare gli anagrammi di una parola
def trova_anagrammi(parola):
    if len(parola) <= 1:
        return [parola]
    else:
        anagrammi = []
        for i in range(len(parola)):
            carattere_corrente = parola[i]
            rimanente = parola[:i] + parola[i + 1:]
            permutazioni_rimanenti = trova_anagrammi(rimanente)
            for permutazione in permutazioni_rimanenti:
                anagrammi.append(carattere_corrente + permutazione)
        return anagrammi


# Chiede all'utente di inserire una parola
parola_input = input("Inserisci una parola: ").strip().lower()
```

```python
# Verifica se la parola ha almeno 2 caratteri
if len(parola_input) < 2:
    print("Inserisci una parola con almeno 2 caratteri.")
else:
    # Chiama la funzione trova_anagrammi
    anagrammi = trova_anagrammi(parola_input)

    # Inizializza una variabile per contare gli anagrammi
    k = 0

    # Stampa gli anagrammi e conta quelli diversi dalla parola di input
    for elemento in anagrammi:
        if elemento != parola_input:
            k += 1
            print(elemento)

    # Stampa il numero totale di anagrammi
    print(f"Gli anagrammi di '{parola_input}' sono: {k}")
```

```
Benvenuto nel Risolutore di Anagrammi!
Inserisci una parola: ciao
cioa
caio
caoi
coia
coai
icao
icoa
iaco
iaoc
ioca
ioac
acio
acoi
aico
aioc
aoci
aoic
ocia
ocai
oica
oiac
oaci
oaic
Gli anagrammi di 'ciao' sono: 23
```

2.2.8 Conteggio delle Lettere nella Frase

[5]:
```
# 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␣
 ↪almeno una volta
    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: ciao
a: 1
c: 1
i: 1
o: 1
```

2.2.9 Convertitore di Valuta con Gestione delle Eccezioni

[6]:
```
# 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
try:
    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"{importo} {valuta_di_partenza} sono equivalenti a␣
↪{importo_convertito:.2f} {valuta_destinazione}")
    else:
        print("Valute non supportate. Assicurati di inserire valute valide.")
except ValueError:
    print("Inserisci un importo valido.")
```

```
Inserisci l'importo da convertire: 400
Inserisci la valuta di partenza: dollari
Inserisci la valuta di destinazione: euro
400.0 dollari sono equivalenti a 340.00 euro
```

[7]:
```
tassi_di_cambio["euro"]
```

[7]:
```
0.85
```

2.2.10 Conteggio dei Prodotti

[8]:
```
conteggio_lettere.items()
```

[8]:
```
dict_items([('a', 1), ('c', 1), ('i', 1), ('o', 1)])
```

[9]:
```
prodotti={}
prodotti["pan bauletto"]=2
prodotti["coca cola"]=3
```

[10]:
```
prodottidue={
    "pan bauletto":2,
    "coca cola":3

}
```

[11]:
```
prodottidue
```

[11]:
```
{'pan bauletto': 2, 'coca cola': 3}
```

2.2.11 Orologio Mondiale

[1]:
```
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): ").strip()
    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):
Londra
L'ora corrente a Londra è: 09:36:21

Città disponibili:
New York
```

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

2.3 Dizionari e Main

2.3.1 Esempio di Funzione Principale con Condizione `if __name__ == "__main__":`

[2]:
```
# Funzione principale può avere qualsiasi nome
def paolo():
    print("mi chiamo paolo")

if __name__ == "__main__": #è una condizione logica che "si verifica sempre" e␣
↪pertanto tutto ciò
                           #che risulta indentato a questa condizione viene␣
↪eseguit
    paolo()
```

```
mi chiamo paolo
```

[3]:
```
# Funzione principale può avere qualsiasi nome
def main():
    print("la funzione principale del codice è stata eseguita, in questa␣
↪funzione possono essere presenti funzioni secondarie \nprecedentemente create")


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

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

2.3.2 Calcolatrice BMI con Funzione Principale

[6]:
```
#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: 62
Inserisci la tua altezza in metri: 1.82
Il tuo BMI è 18.72, sei classificato come 'Normopeso'.
```

[7]:
```
# 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: 6
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 62
Inserisci la tua altezza in metri: 1.82
Il tuo BMI è 18.72, sei classificato come 'Normopeso'.
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 456
Inserisci la tua altezza in metri: 345
Il tuo BMI è 0.00, sei classificato come 'Sottopeso'.
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 345
```

```
Inserisci la tua altezza in metri: 345
Il tuo BMI è 0.00, sei classificato come 'Sottopeso'.
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 4
Inserisci la tua altezza in metri: 5
Il tuo BMI è 0.16, sei classificato come 'Sottopeso'.
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 6
Inserisci la tua altezza in metri: 67
Il tuo BMI è 0.00, sei classificato come 'Sottopeso'.
Benvenuto nella Calcolatrice BMI!
Inserisci il tuo peso in chilogrammi: 4
Inserisci la tua altezza in metri: 5
Il tuo BMI è 0.16, sei classificato come 'Sottopeso'.
```

2.3.3 Convertitore di Unità di Misura con Funzione Principale e Selezione

[8]:
```python
#### 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):␣
↪").lower()
    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: 45
 45.000 metri corrispondono a  147.638 piedi.
```

2.3.4 Registro Alimentare con Funzione Principale e Calcolo Calorie

[44]:
```
# Dizionario con le calorie per 100 grammi di cibo
cibo_calorie = {
    "banana": 89,
    "mela": 52,
    "arancia": 43,
    # Altri cibi...
}

# Funzione per calcolare le calorie consumate
def calorie_consumate(cibo, quantita):
    if cibo not in cibo_calorie:
        print("Cibo non presente")
        return 0  # Ritorna 0 calorie se il cibo non è nel dizionario
    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("\n1. Aggiungi cibo consumato")
        print("2. Calcola calorie totali")
        print("3. Esci")
```

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

        if scelta == "1":
            print("\nCibi disponibili:")
            for cibo in cibo_calorie:
                print(cibo)
            cibo = input("Inserisci il cibo consumato: ").lower()
            quantita = float(input("Inserisci la quantita (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
```

[9]:
```
acquisti={}
acquisti["pan bauletto"]=10
acquisti["nutella"]=10
```

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

}
```

[11]:
```
acquistidue
```

[11]:
```
{'pan bauletto': 10, 'nutella': 10}
```

2.4 Generatore di Personaggi

2.4.1 Generatore Casuale di Personaggi Fantasy

[17]:
```
# Importa il modulo random per generare valori casuali
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", "Incantesimi di guarigione"]

# 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))  # Sceglie␣
 ↪casualmente da 1 a 3 abilità
```

```
# Stampa il personaggio generato
print("Personaggio Fantasy Generato:")
print(f"Specie: {specie}")
print(f"Classe: {classe}")
print(f"Arma: {arma}")
print(f"Abilità: {', '.join(abilita_scelte)}")  # Converte la lista di abilità␣
 ↪in una stringa separata da virgole
```

```
Personaggio Fantasy Generato:
Specie: Umano
Classe: Guerriero
Arma: Arco
Abilità: Incantesimi di guarigione, Furtività
```

2.4.2 Generatore Casuale di Personaggi Fantasy (con Funzione)

[53]:
```
import random

# Liste di speci, 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", "Incantesimi di guarigione"]

# 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
```

2.4.3 Generatore di Personaggi per Romanzi

[22]:
```python
import random

# Liste di tratti fisici, tratti di personalità, sfondi e motivazioni
physical_traits = ["capelli neri", "capelli biondi", "occhi azzurri", "occhi␣
 ↪verdi", "pelle chiara", "pelle scura", "alto", "basso", "atletico",␣
 ↪"sovrappeso"]
personality_traits = ["gentile", "introverso", "estroverso", "ottimista",␣
 ↪"pessimista", "ambizioso", "timido", "curioso", "spiritoso", "serio"]
backgrounds = ["contadino", "nobile", "artigiano", "commerciante",␣
 ↪"avventuriero", "scienziato", "musicista", "insegnante", "guerriero", "poeta"]
motivations = ["vendetta", "ricchezza", "amore", "vita eterna", "conoscenza",␣
 ↪"fama", "avventura", "pace", "giustizia", "libertà"]

# Funzione per generare un personaggio casuale
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

# Stampa il risultato
print("Generatore di Personaggi per Romanzi")
print(genera_personaggio())
```

```
Generatore di Personaggi per Romanzi
Inserisci il nome del personaggio: Mattia
Nome: Mattia
Aspetto fisico: pelle chiara
Aspetto personale: pessimista
Sfondo: guerriero
Motivazione: pace
```

2.5 La Letteratura Combinatoria

2.5.1 Generatore di Titoli Nobiliari

[25]:
```
import random

# Database di citazioni
citazioni = [
    "La vita è ciò che succede mentre sei occupato a fare altri progetti. - John␣
↪Lennon",
    "Il successo è camminare da un fallimento all'altro senza perdere␣
↪l'entusiasmo. - Winston Churchill",
    "La felicità è quando ciò che pensi, ciò che dici e ciò che fai sono in␣
↪armonia. - Mahatma Gandhi",
    "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: {citazione}")

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

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

2.5.2 Generatore di Post da Influencer

[26]:
```
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 da utilizzare
    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:
Rischiare è vivere. Trova la tua passione. La saggezza del fallimento. Sii
creativo. Inseguire i tuoi sogni. Il potere della positività. Fai la differenza.
```

2.5.3 Generatore di Citazioni Personalizzate

[29]:
```python
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 temono.",
    "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 nascosta.",
    "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 che␣
 ↪hanno.",
    "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 da utilizzare
    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:
La vita è troppo importante per essere presa sul serio. Cambia il mondo. La
saggezza del fallimento. Rischiare è vivere.
```

2.5.4 Generatore di Poesie Casuali

[28]:
```python
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())
```

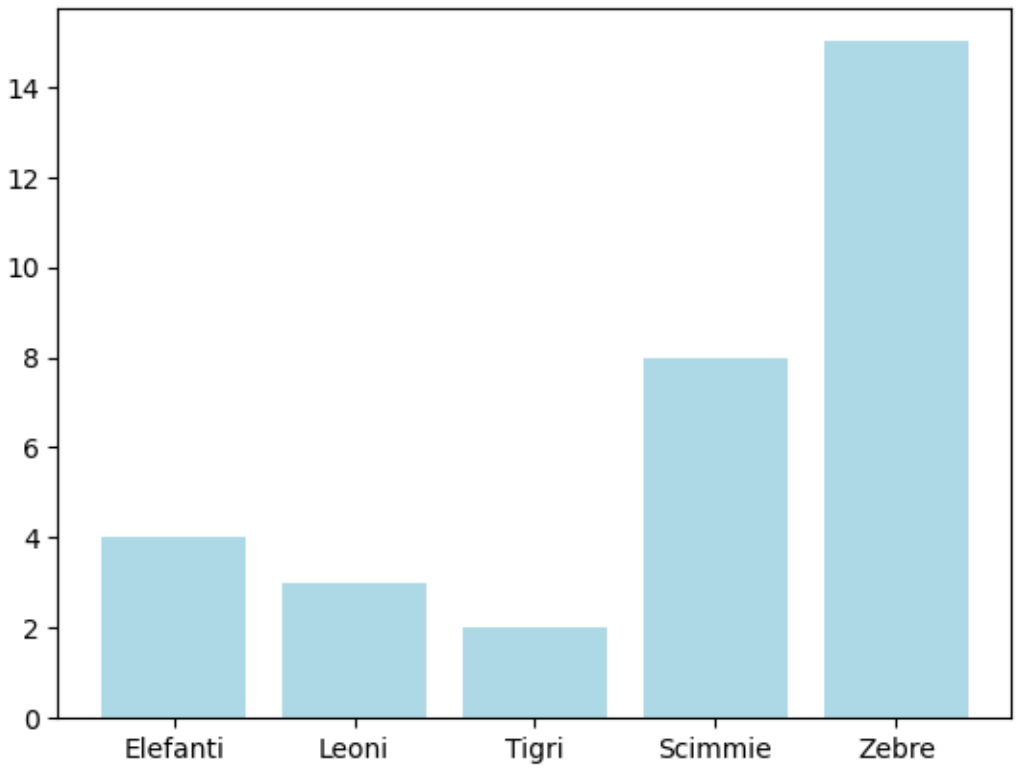
```
Il profondo vento danza.
Il luminoso vento canta.
Nel vento canta con sereno mare.
```

3 ESERCITAZIONE 3

3.1 Grafici

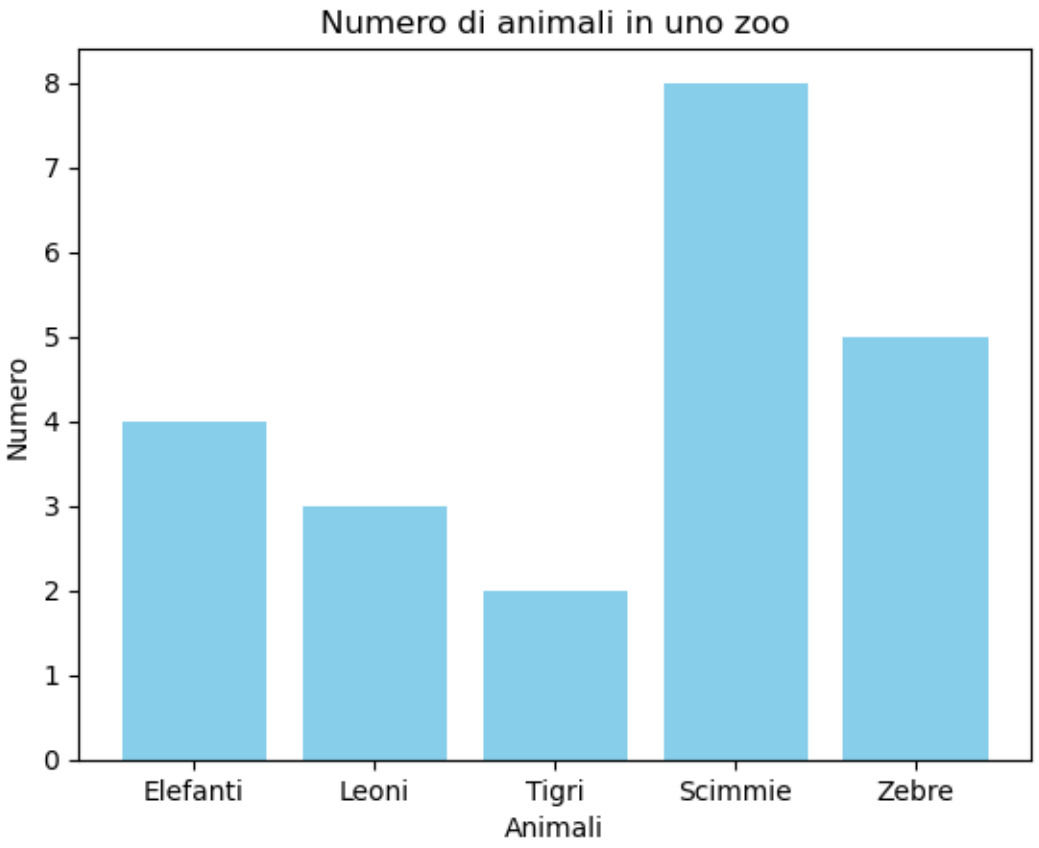
3.1.1 Grafico a Barre della Distribuzione del Numero di Animali nello Zoo

[137]:
```python
# Codice per generare un diagramma a barre che visualizza la distribuzione del␣
 ↪numero di animali in uno zoo
import matplotlib.pyplot as plt

# Dati relativi al numero di animali
animali = ['Elefanti', 'Leoni', 'Tigri', 'Scimmie', 'Zebre']
numero_animali = [4, 3, 2, 8, 15]

# Creazione del diagramma a barre con colorazione chiara
plt.bar(animali, numero_animali, color="lightblue")

# Visualizzazione del diagramma
plt.show()
```



3.1.2 Grafico a Barre della Distribuzione del Numero di Animali nello Zoo

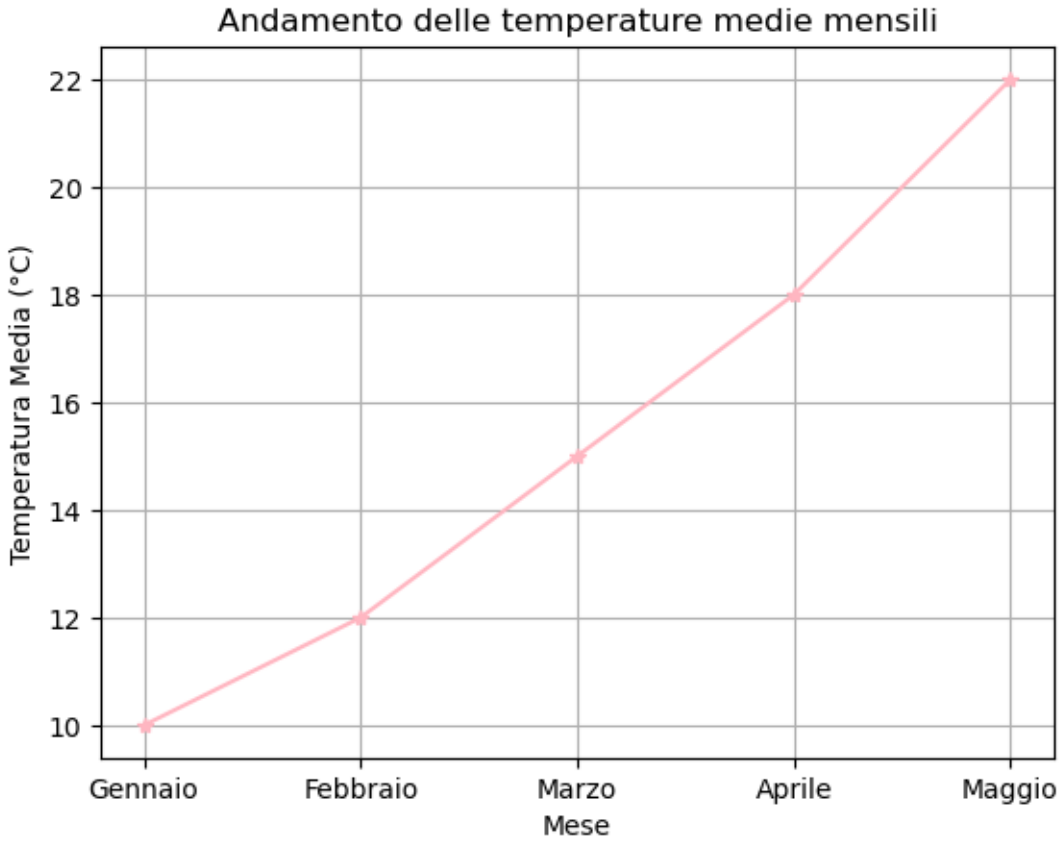
[4]:
```
# Codice per generare un grafico a barre che visualizza la distribuzione del␣
↪numero di animali in uno zoo
import matplotlib.pyplot as plt

# Dati relativi al numero di animali
animali = ['Elefanti', 'Leoni', 'Tigri', 'Scimmie', 'Zebre']
numero_animali = [4, 3, 2, 8, 5]

# Creazione del grafico a barre con colorazione chiara
plt.bar(animali, numero_animali, color='skyblue')

# Aggiunta di titolo e etichette agli assi
plt.title('Numero di animali in uno zoo')
plt.xlabel('Animali')
plt.ylabel('Numero')

# Visualizzazione del grafico
plt.show()
```



3.1.3 Grafico a Linee del Andamento delle Temperature Medie Mensili nel Corso dell'Anno

[5]:
```
# Codice per generare un grafico a linee che visualizza l'andamento delle␣
↪temperature medie mensili
import matplotlib.pyplot as plt

# Dati relativi alle temperature medie mensili
mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]

# Creazione del grafico a linee con marcatori a forma di stella e linea di␣
↪connessione
plt.plot(mese, temperatura_media, marker='*', linestyle='-', color='lightpink')

# Aggiunta di titolo e etichette agli assi
plt.title('Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')
```

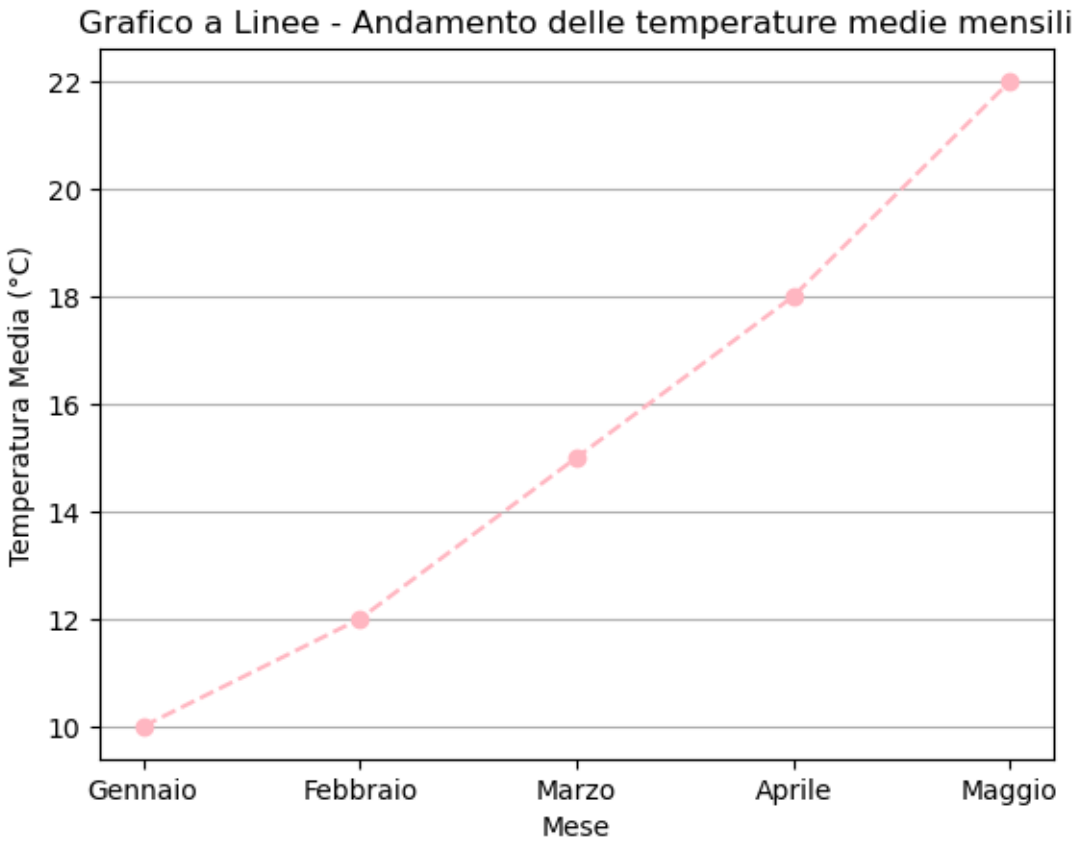
```
# Attivazione delle linee guida sulla griglia per una migliore lettura
plt.grid(True)

# Visualizzazione del grafico
plt.show()
```



3.1.4 Grafico a Linee del Andamento delle Temperature Medie Mensili nel Corso dell'Anno

[8]:
```
# Codice per generare un grafico a linee che visualizza l'andamento delle␣
↪temperature medie mensili
import matplotlib.pyplot as plt

# Dati relativi alle temperature medie mensili
mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
temperatura_media = [10, 12, 15, 18, 22]

# Creazione del grafico a linee con marcatori circolari e stile tratteggiato
plt.plot(mese, temperatura_media, marker='o', linestyle='--', color='lightpink')
```

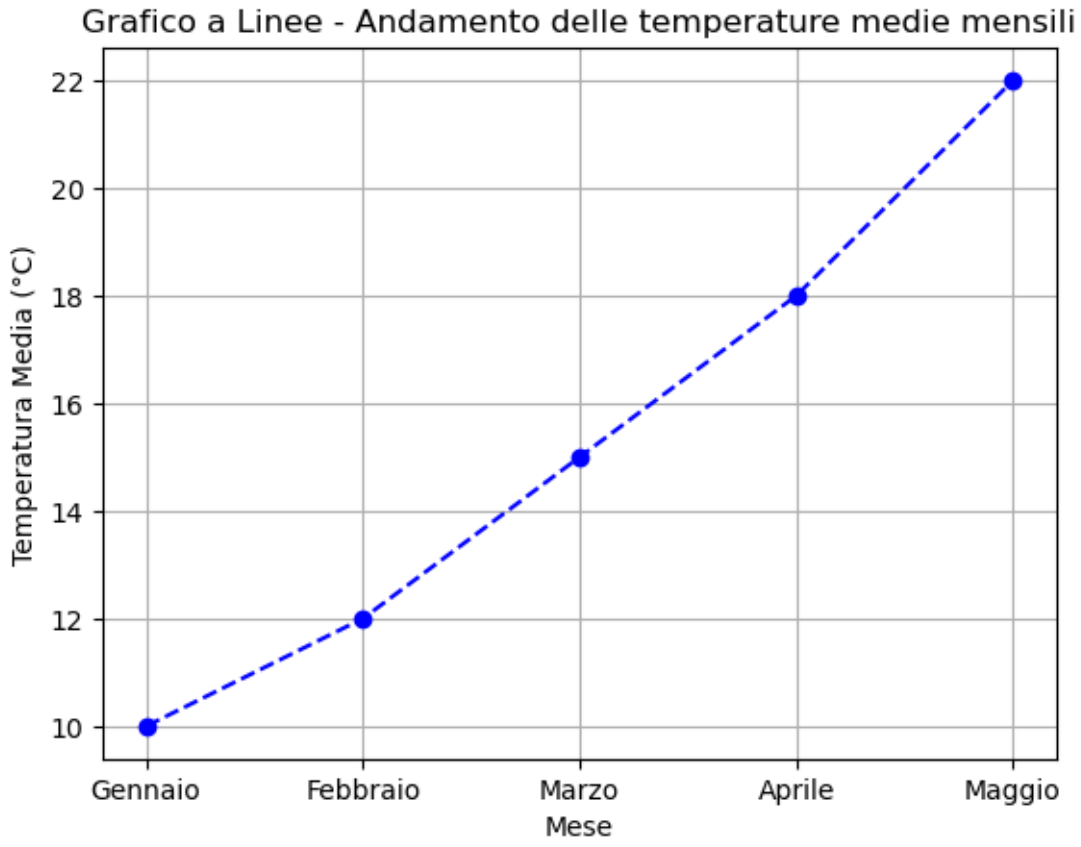
```
# Aggiunta di titolo e etichette agli assi
plt.title('Grafico a Linee - Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')

# Attivazione delle linee guida solo sull'asse y per una migliore lettura
plt.grid(True, axis="y")

# Visualizzazione del grafico
plt.show()
```



3.1.5 Grafico a Linee del Andamento delle Temperature Medie Mensili nel Corso dell'Anno

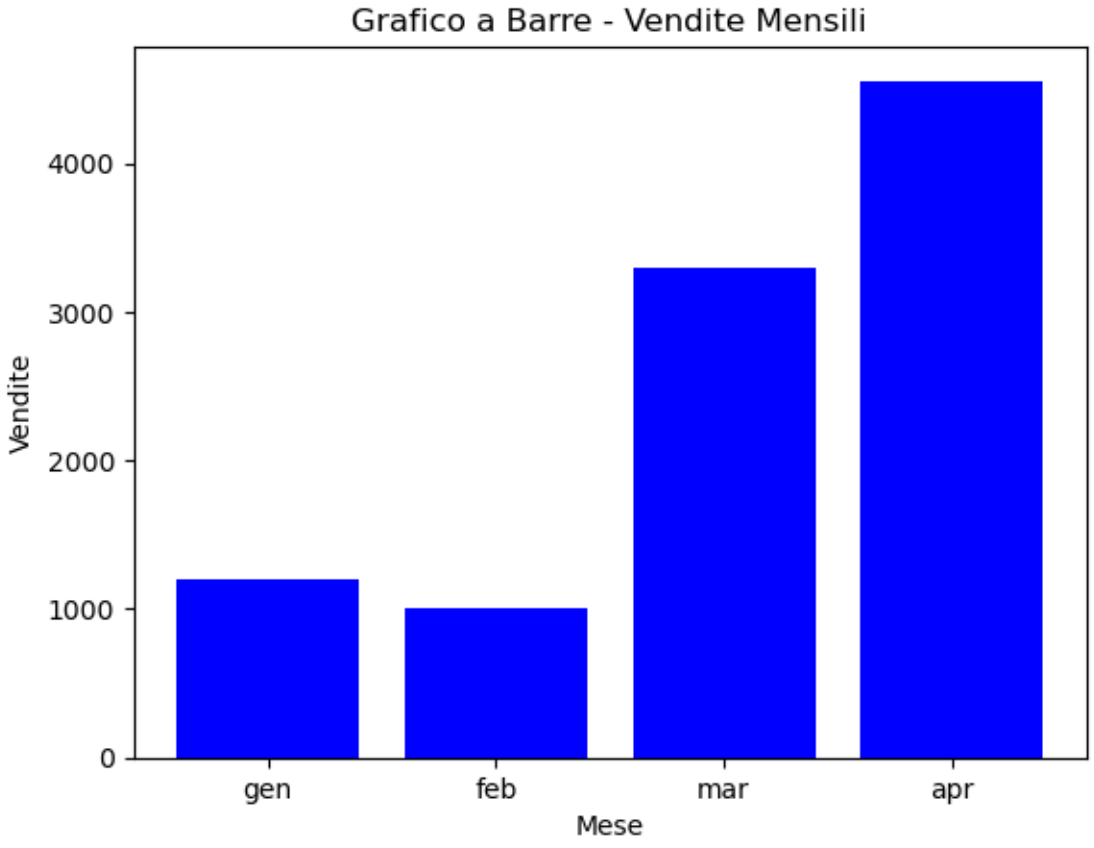
[9]:
```
# Codice per generare un grafico a linee che visualizza l'andamento delle␣
↪temperature medie mensili
import matplotlib.pyplot as plt

# Dati relativi alle temperature medie mensili
mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
```

```
temperatura_media = [10, 12, 15, 18, 22]

# Creazione del grafico a linee con marcatori circolari, stile tratteggiato e␣
 ↪colore blu
plt.plot(mese, temperatura_media, marker='o', linestyle='--', color='blue')

# Aggiunta di titolo e etichette agli assi
plt.title('Grafico a Linee - Andamento delle temperature medie mensili')
plt.xlabel('Mese')
plt.ylabel('Temperatura Media (°C)')

# Attivazione delle linee guida sulla griglia per una migliore lettura
plt.grid(True)

# Visualizzazione del grafico
plt.show()
```



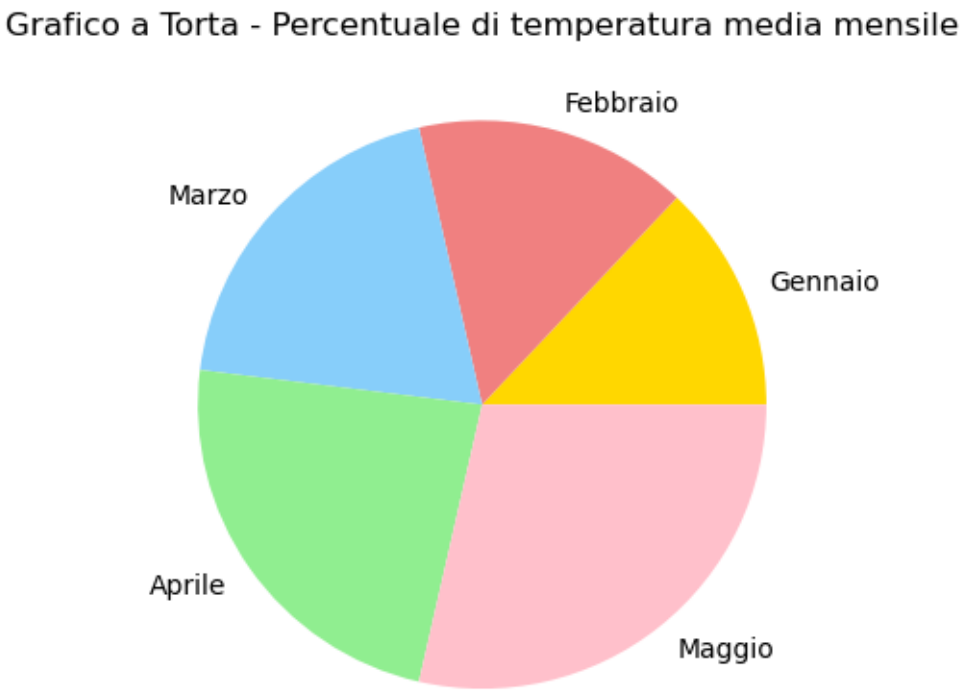
3.1.6 Grafico a Barre delle Vendite Mensili

[10]:
```
# Codice per generare un grafico a barre che visualizza le vendite mensili
import matplotlib.pyplot as plt

# Dati relativi alle vendite mensili
vendite_mensili = {
    "gen": 1200,
    "feb": 1000,
    "mar": 3300,
    "apr": 4555
}

# Creazione del grafico a barre con colore blu
plt.bar(vendite_mensili.keys(), vendite_mensili.values(), color="blue")

# Aggiunta di titolo e etichette agli assi
plt.title('Grafico a Barre - Vendite Mensili')
plt.xlabel('Mese')
plt.ylabel('Vendite')

# Visualizzazione del grafico
plt.show()
```



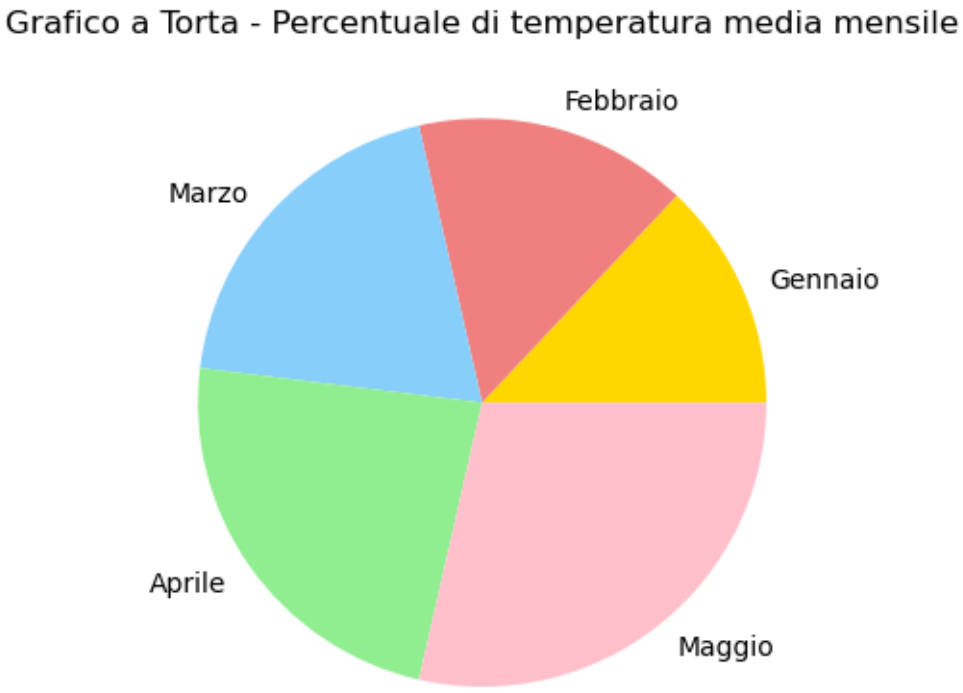
3.1.7 Grafico a Torta della Percentuale di Temperatura Media Mensile

[11]:
```
# Codice per generare un grafico a torta che mostra la percentuale di␣
↪temperatura media mensile
import matplotlib.pyplot as plt

# Dati relativi alle temperature medie mensili, colori e mesi
temperatura_media = [10, 12, 15, 18, 22]
colori = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']
mese = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']

# Creazione del grafico a torta con etichette dei mesi e colori specifici
plt.pie(temperatura_media, labels=mese, colors=colori)

# Aggiunta di titolo al grafico
plt.title('Grafico a Torta - Percentuale di temperatura media mensile')

# Visualizzazione del grafico
plt.show()
```



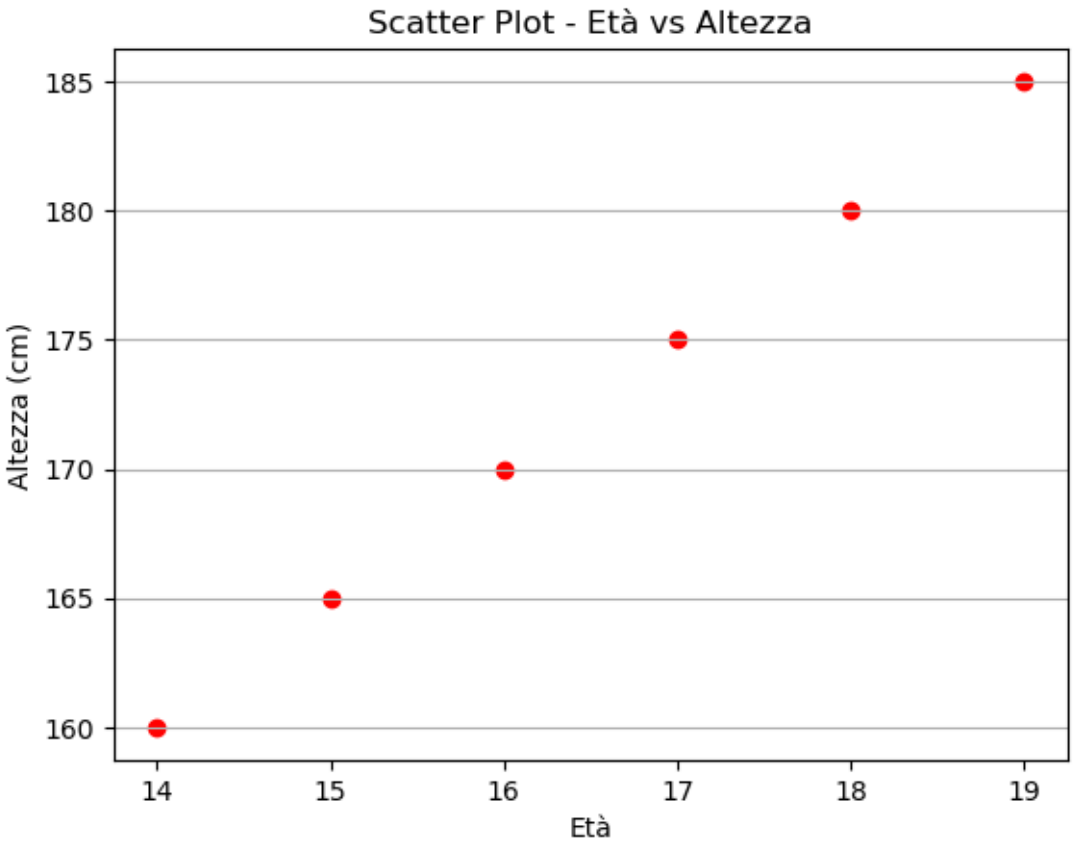
3.1.8 Grafico a Torta della Percentuale di Temperatura Media Mensile

[12]:
```
# Codice per generare un grafico a torta che mostra la percentuale di␣
↪temperatura media mensile
import matplotlib.pyplot as plt

# Dati relativi alle temperature medie mensili, colori e nomi dei mesi
temperatura_mesi = {
    'Gennaio': 10,
    'Febbraio': 12,
    'Marzo': 15,
    'Aprile': 18,
    'Maggio': 22
}
colori = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink']

# Creazione del grafico a torta con etichette dei mesi e colori specifici
plt.pie(temperatura_mesi.values(), labels=temperatura_mesi.keys(), colors=colori)

# Aggiunta di titolo al grafico
plt.title('Grafico a Torta - Percentuale di temperatura media mensile')

# Visualizzazione del grafico
plt.show()
```



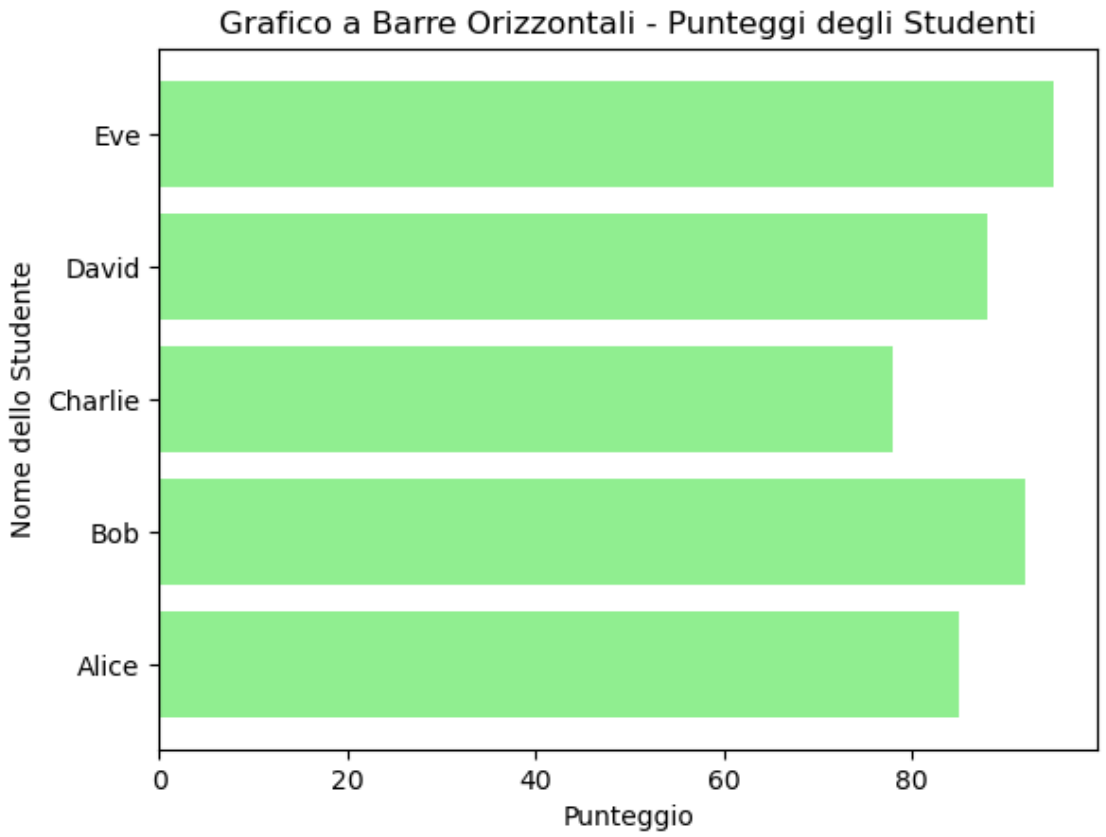
3.1.9 Scatter Plot del'Età vs Altezza

[13]:
```
# Codice per generare uno scatter plot che mostra la relazione tra età e altezza
import matplotlib.pyplot as plt

# Dati relativi a età e altezza
età = [14, 15, 16, 17, 18, 19]
altezza = [160, 165, 170, 175, 180, 185]

# Creazione dello scatter plot con cerchi rossi
plt.scatter(età, altezza, color='red', marker='o')#In questo caso marker, 'o'␣
↪indica l'uso di cerchi come marcatori

# Aggiunta di titolo e etichette agli assi
plt.title('Scatter Plot - Età vs Altezza')
plt.xlabel('Età')
plt.ylabel('Altezza (cm)')

# Attivazione delle linee guida sulla griglia sull'asse y per una migliore␣
↪lettura
plt.grid(True, axis='y')

# Visualizzazione dello scatter plot
plt.show()
```



3.1.10 Grafico a Barre Orizzontali dei Punteggi degli Studenti

[15]:
```
# Codice per generare un grafico a barre orizzontali che mostra i punteggi degli␣
↪studenti
import matplotlib.pyplot as plt

# Dati relativi ai nomi degli studenti e ai loro punteggi
nomi_studenti = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
punteggi = [85, 92, 78, 88, 95]

# Creazione del grafico a barre orizzontali con colori verdi chiari
plt.barh(nomi_studenti, punteggi, color='lightgreen')

# Aggiunta di titolo e etichette agli assi
plt.title('Grafico a Barre Orizzontali - Punteggi degli Studenti')
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')

# Visualizzazione del grafico
plt.show()
```



3.1.11 DataFrame dei Punteggi degli Studenti (Ordinati per Punteggio)

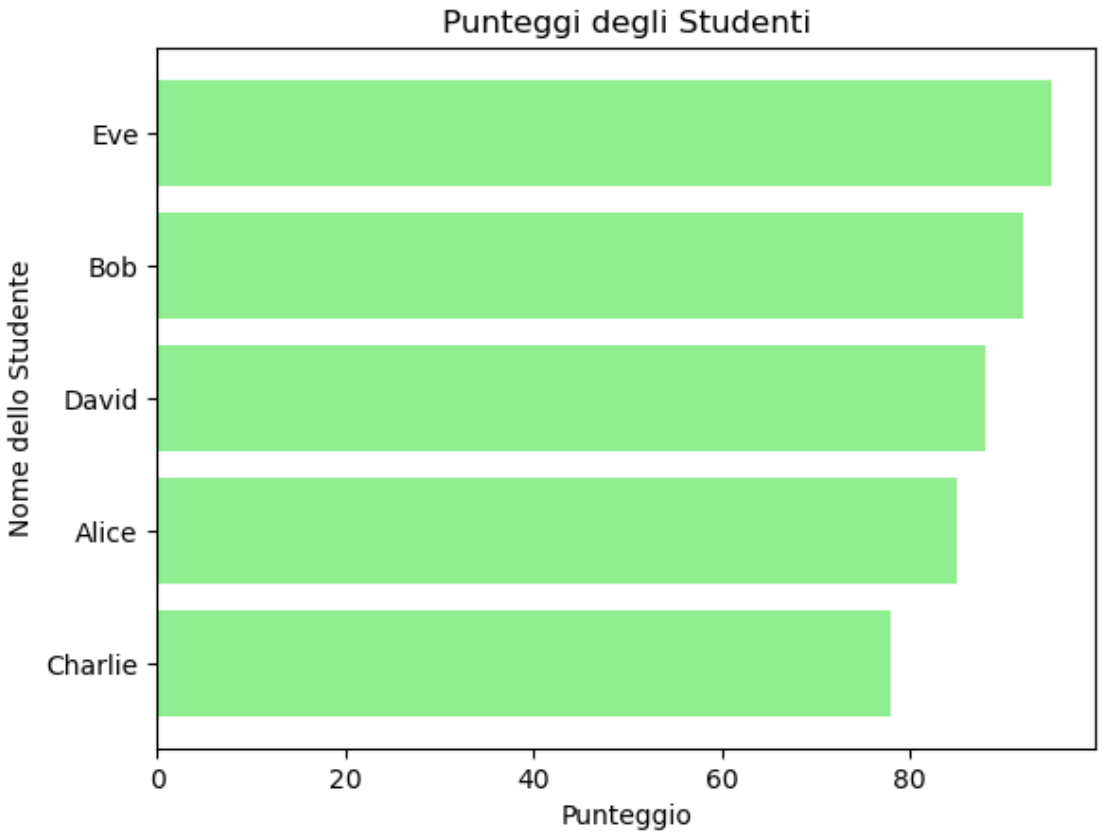
[10]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Dati relativi ai nomi degli studenti e ai loro punteggi
nomi_studenti = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
punteggi = [85, 92, 78, 88, 95]

# Crea un DataFrame utilizzando pandas
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)

# Visualizza il DataFrame ordinato
df
```

[10]:

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

3.1.12 Grafico a Barre Orizzontali dei Punteggi degli Studenti

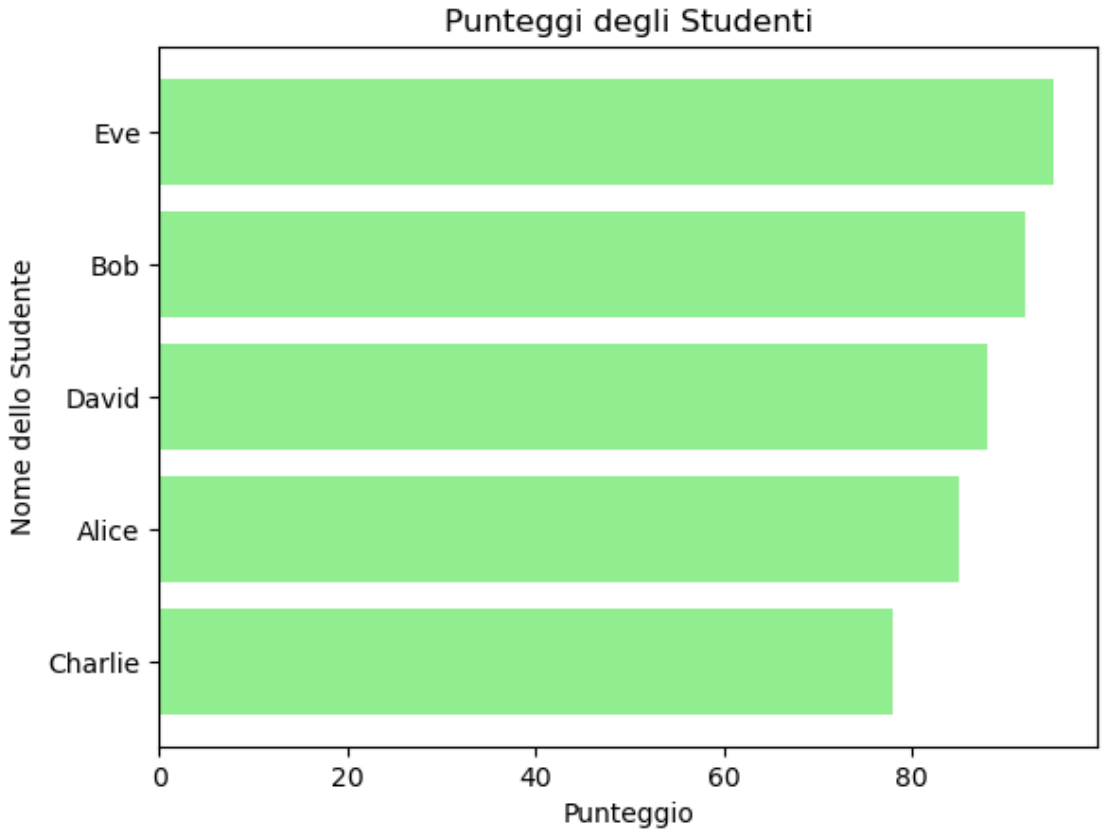
[19]:
```
# Crea un grafico a barre orizzontali utilizzando i dati dal DataFrame ordinato
plt.barh(df['Nome dello Studente'], df['Punteggio'], color='lightgreen')

# Aggiunge un titolo al grafico
plt.title('Punteggi degli Studenti')

# Etichette degli assi x e y
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')

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



3.1.13 Grafico a Barre Orizzontali dei Punteggi degli Studenti

[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 e crea un nuovo␣
↪DataFrame ordinato
df_sorted = df.sort_values(by='Punteggio')

# Crea un grafico a barre orizzontali utilizzando i dati dal DataFrame ordinato
plt.barh(df_sorted['Nome dello Studente'], df_sorted['Punteggio'],␣
↪color='lightgreen')
```

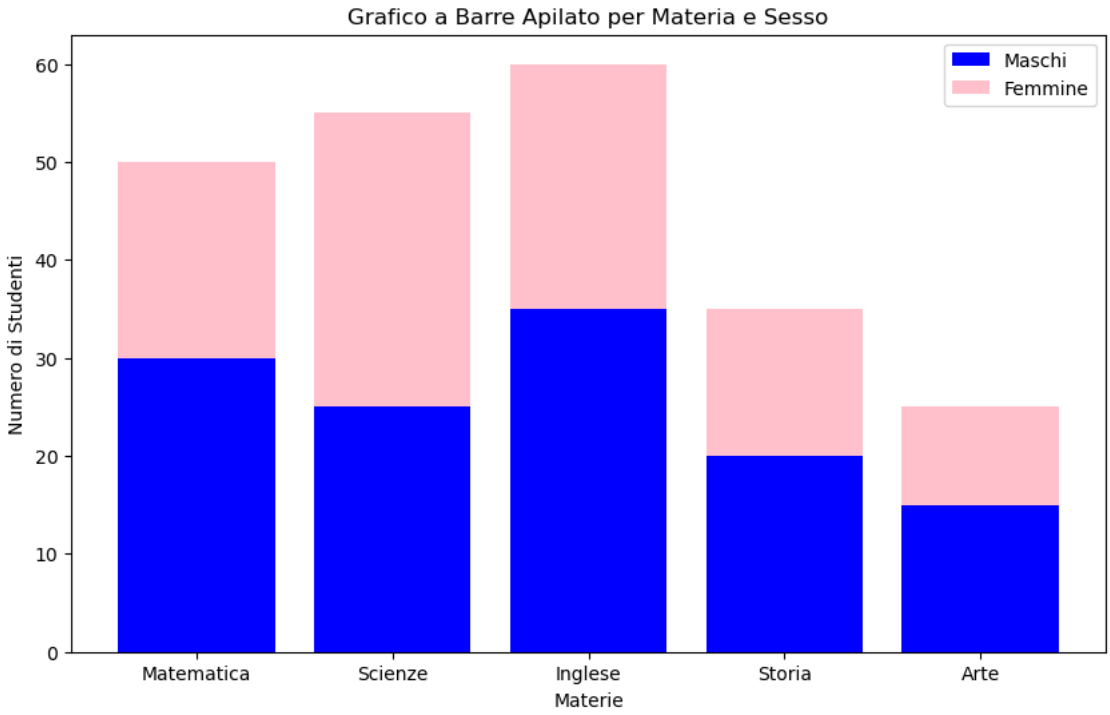
```
# Aggiunge un titolo al grafico
plt.title('Punteggi degli Studenti')

# Etichette degli assi x e y
plt.xlabel('Punteggio')
plt.ylabel('Nome dello Studente')

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



3.1.14 Grafico a Barre Apilato per Materia e Sesso

[18]:
```
# 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')
```

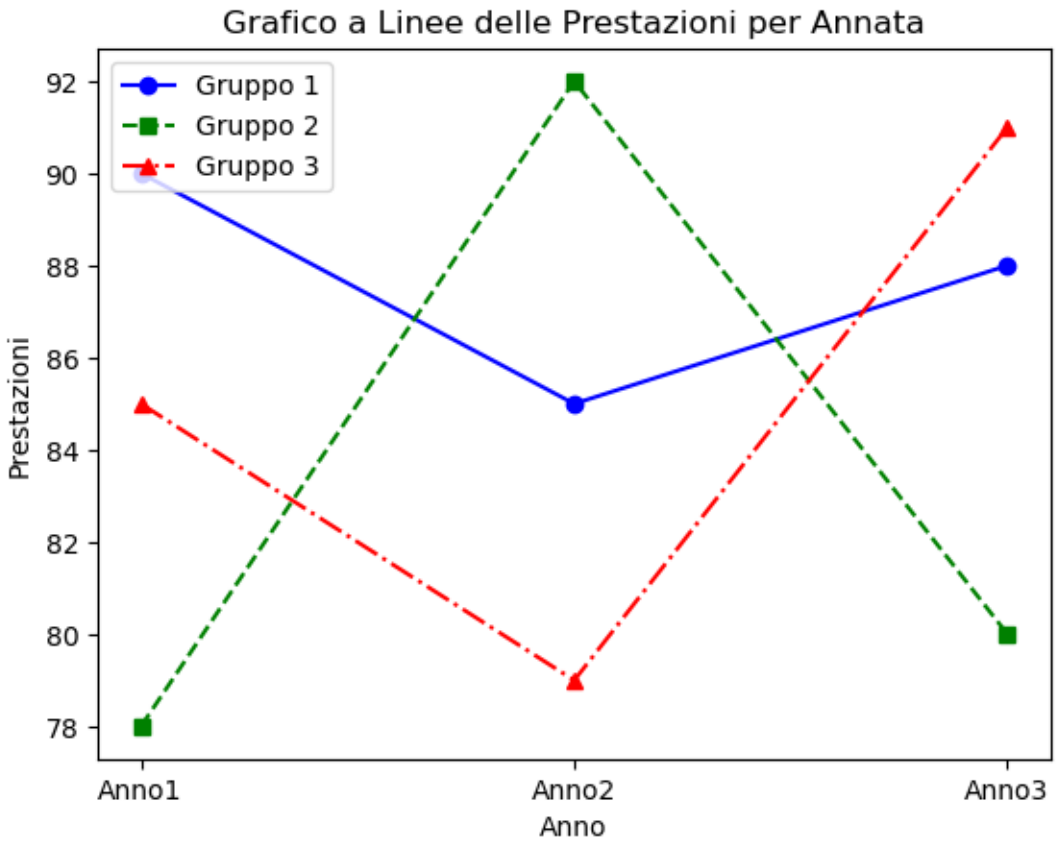
```
plt.bar(materie, femmine, label='Femmine', bottom=maschi, color='pink')

# 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()
```



3.1.15 Grafico a Linee delle Prestazioni per Annata

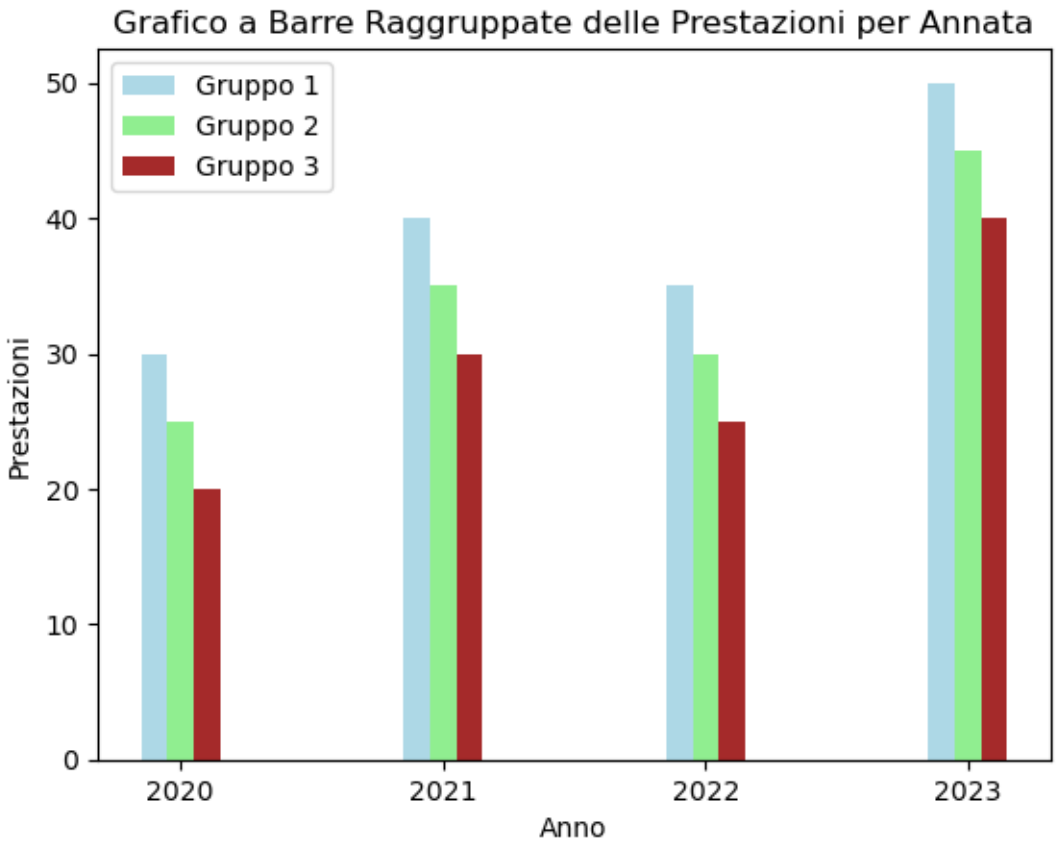
[20]:
```
# Creazione di liste per rappresentare gli anni e le prestazioni dei tre gruppi
annata = ['Anno1', 'Anno2', 'Anno3']
gruppo1 = [90, 85, 88]
gruppo2 = [78, 92, 80]
gruppo3 = [85, 79, 91]

# Creazione di un grafico a linee per visualizzare le prestazioni nel tempo per␣
↪ciascun gruppo
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')

# Aggiunta di titolo e etichette agli assi
plt.title('Grafico a Linee delle Prestazioni per Annata')
plt.xlabel('Anno')
plt.ylabel('Prestazioni')

# Aggiunta della legenda in alto a sinistra per identificare i gruppi nel grafico
plt.legend(loc='upper left')

# Visualizzazione del grafico
plt.show()
```



3.1.16 Grafico a Barre Raggruppate delle Prestazioni dei Gruppi per Anno

[21]:
```
# Importazione delle librerie necessarie per il grafico
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio: annate e prestazioni dei tre gruppi
annata = ['2020', '2021', '2022', '2023']
gruppo1 = [30, 40, 35, 50]
gruppo2 = [25, 35, 30, 45]
gruppo3 = [20, 30, 25, 40]

# Definizione della larghezza delle barre e degli indici per la posizione delle␣
 ↪barre sul grafico
larghezza_barre = 0.1
indici = np.arange(len(annata))#len(annata): Restituisce la lunghezza della␣
 ↪lista annata, cioè il numero di elementi presenti in essa. Nel tuo caso,␣
 ↪rappresenta il numero di anni.np.arange(len(annata)): Utilizza la funzione␣
 ↪arange di NumPy per creare un array di numeri interi da 0 a len(annata) - 1.␣
 ↪Questo array di numeri interi viene assegnato alla variabile indici. Quindi,␣
 ↪se annata contiene ['2020', '2021', '2022', '2023'], len(annata) sarà 4, e np.
 ↪arange(len(annata)) creerà l'array [0, 1, 2, 3]. Questi valori rappresentano␣
 ↪gli indici che vengono utilizzati per posizionare le barre nel grafico in modo␣
 ↪che siano distribuite uniformemente lungo l'asse x per ogni anno nel datase
# Creazione di un grafico a barre raggruppate per visualizzare le prestazioni␣
 ↪dei tre gruppi per ogni anno
plt.bar(indici - larghezza_barre, gruppo1, width=larghezza_barre, label='Gruppo␣
 ↪1', color='lightblue')#indici - larghezza_barre: Posiziona le barre del Gruppo␣
 ↪1 a sinistra rispetto agli indici calcolati in precedenza, ottenendo così una␣
 ↪separazione tra i gruppi di barre.
plt.bar(indici, gruppo2, width=larghezza_barre, label='Gruppo 2',␣
 ↪color='lightgreen')
plt.bar(indici + larghezza_barre, gruppo3, width=larghezza_barre, label='Gruppo␣
 ↪3', color='brown')

# Aggiunta di titolo, etichette degli assi e legenda
plt.title('Grafico a Barre Raggruppate delle Prestazioni per Annata')
plt.xlabel('Anno')
plt.ylabel('Prestazioni')
plt.xticks(indici, annata)  # Etichette degli assi con gli anni
plt.legend(loc='upper left')

# Visualizzazione del grafico
plt.show()
```



[22]:
```
indici = np.arange(len(annata))
```

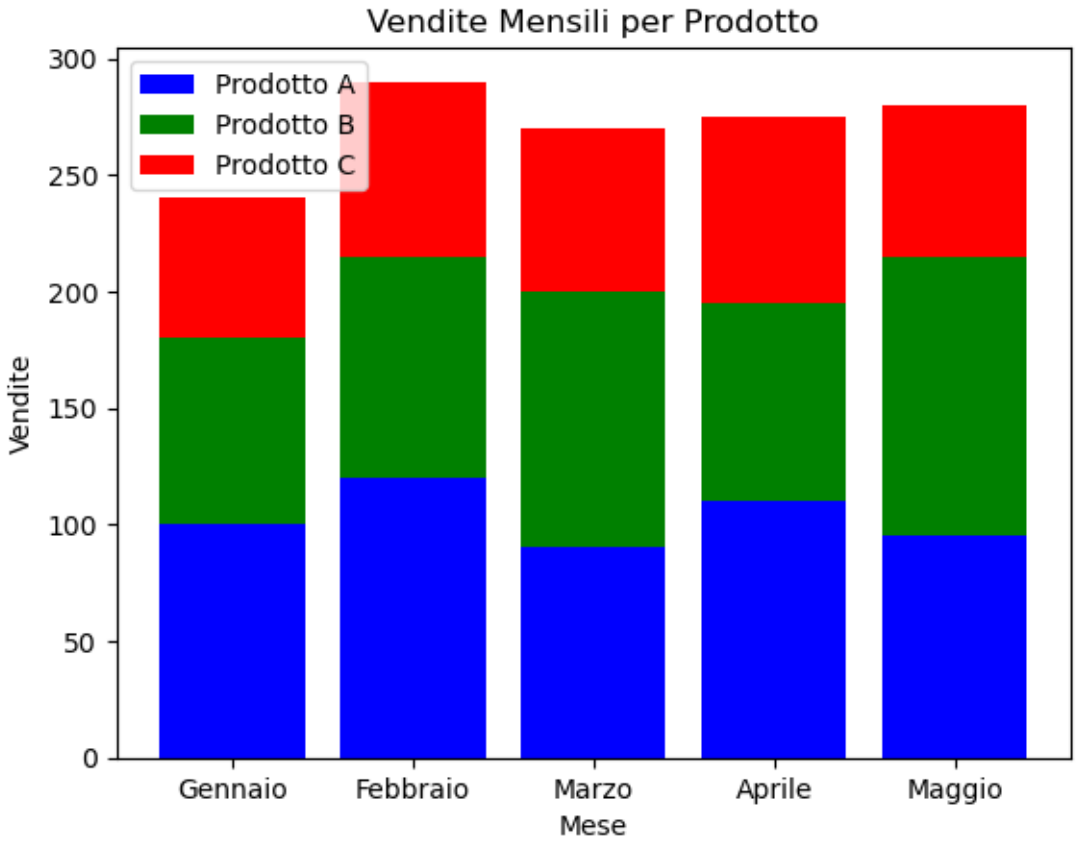
[23]:
```
indici - larghezza_barre
```

[23]:
```
array([-0.1,  0.9,  1.9,  2.9])
```

3.1.17 Grafico a Barre Empilate delle Vendite Mensili per Prodotto

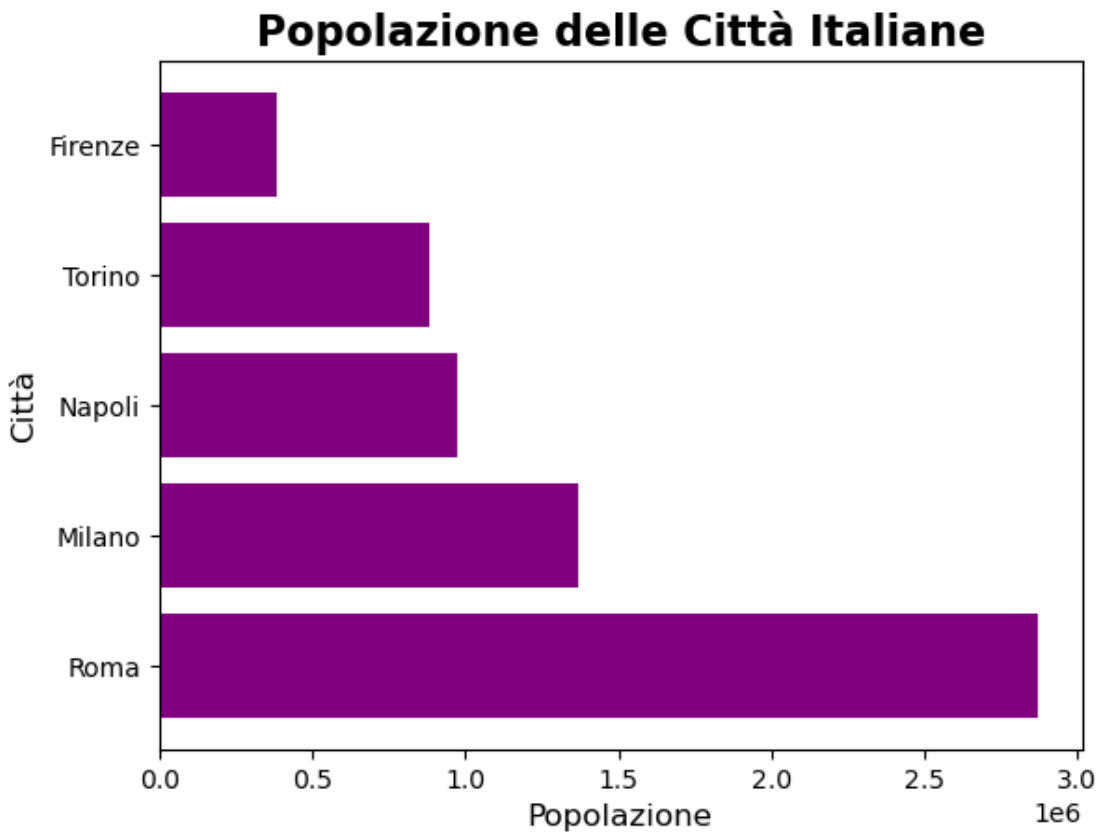
[1]:
```
import matplotlib.pyplot as plt
import numpy as np

# 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') # bottom è␣
 ↪un parametro della funzione plt.bar che determina da dove inizia la pila di␣
 ↪barre empilate
plt.bar(mesi, vendite_prodotto_B, label='Prodotto B', color='green',␣
 ↪bottom=vendite_prodotto_A) # Le barre del "Prodotto B" iniziano dalla cima␣
 ↪delle barre del "Prodotto A", quindi la pila si sviluppa sopra le barre del␣
 ↪"Prodotto A".
plt.bar(mesi, vendite_prodotto_C, label='Prodotto C', color='red', bottom=np.
 ↪array(vendite_prodotto_A) + np.array(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()
```



3.1.18 Grafico a Barre Orizzontali della Popolazione delle Città Italiane

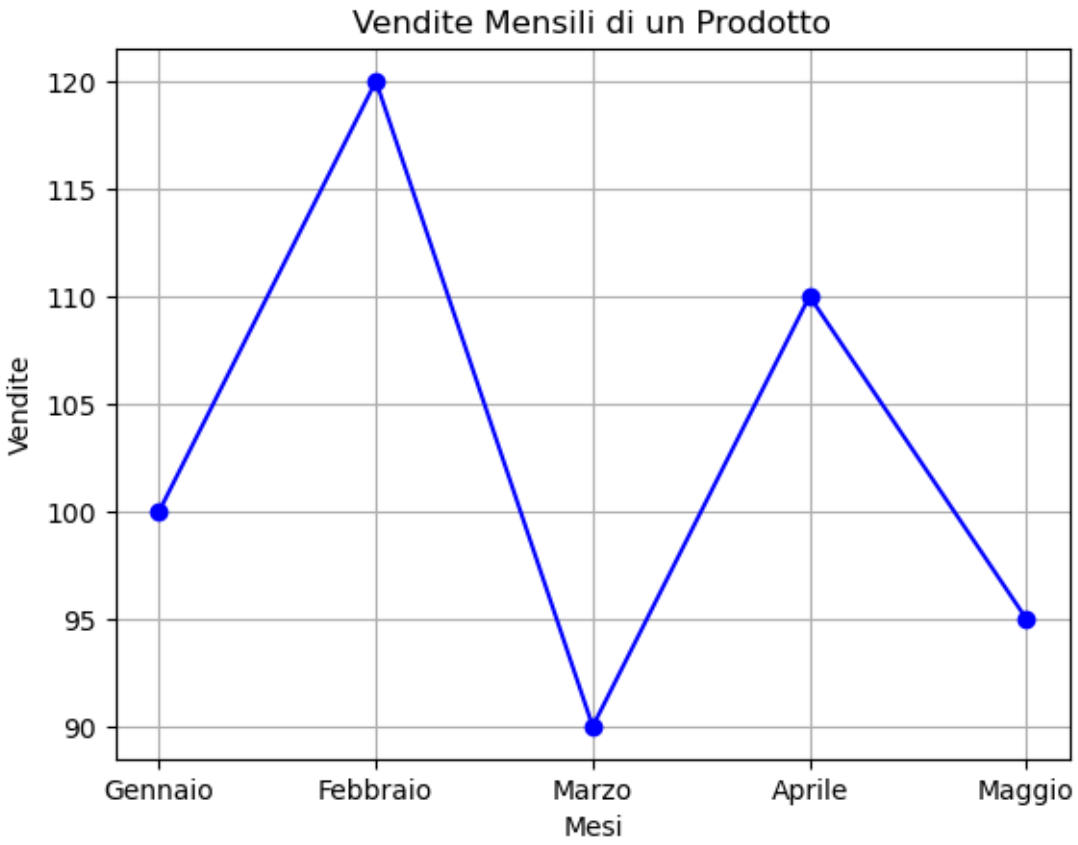
[31]:
```
# Importa le librerie necessarie per il grafico
import matplotlib.pyplot as plt
import numpy as np

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

# Crea un grafico a barre orizzontali
plt.barh(città, popolazione, color='purple')

# Aggiungi un titolo al grafico
plt.title('Popolazione delle Città Italiane', fontsize=16, fontweight='bold')

# Etichetta l'asse x
plt.xlabel('Popolazione', fontsize=12)

# Etichetta l'asse y
plt.ylabel('Città', fontsize=12)

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



3.1.19 Grafico a Linee dell'Andamento Mensile

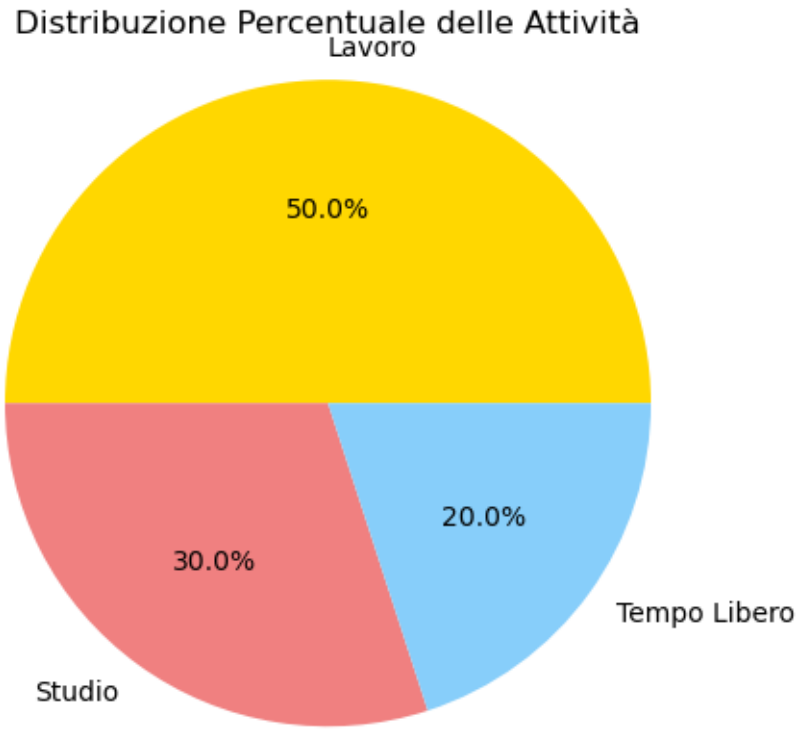
[32]:
```
# Importa le librerie necessarie per il grafico
import matplotlib.pyplot as plt
import numpy as np

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

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

# Personalizza il grafico
plt.title('Vendite Mensili di un Prodotto')  # Titolo del grafico
plt.xlabel('Mesi')  # Etichetta dell'asse x
plt.ylabel('Vendite')  # Etichetta dell'asse y
plt.grid(True)  # Mostra la griglia nel grafico

# Mostra il grafico risultante
```

```
plt.show()
```



3.1.20 Grafico a Torta della Distribuzione Percentuale delle Attività

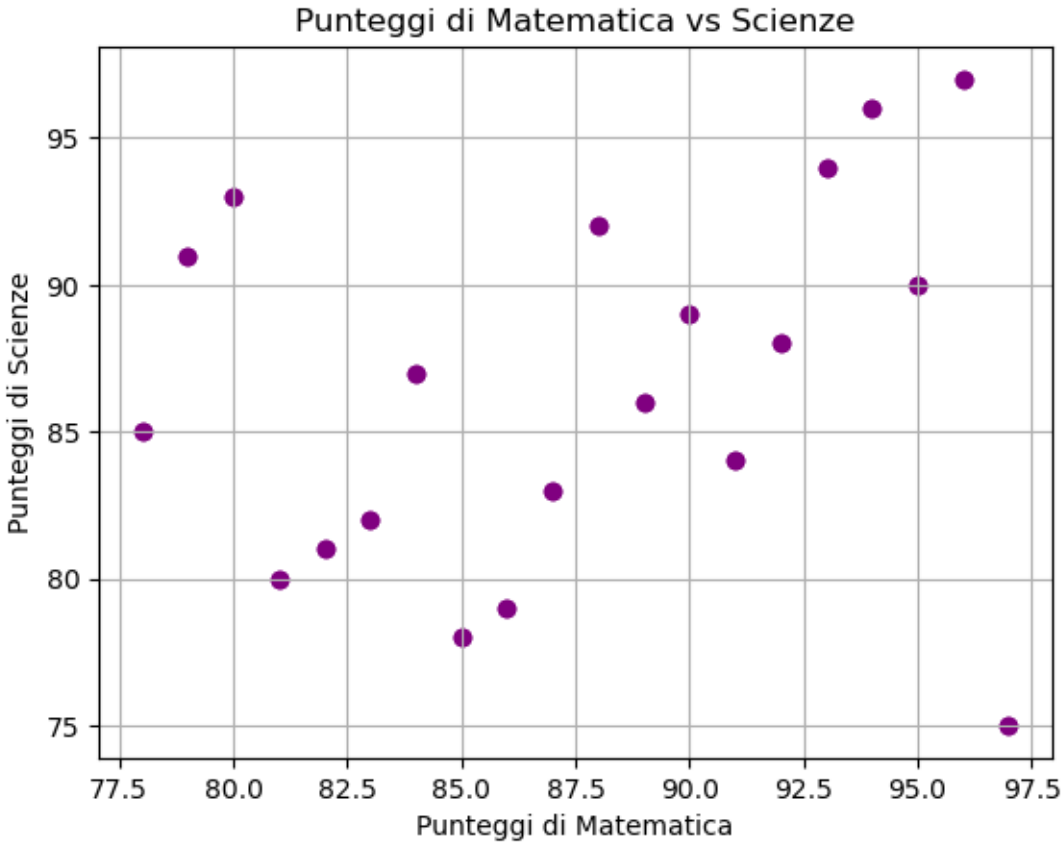
[6]:
```
import matplotlib.pyplot as plt

# 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('Distribuzione Percentuale delle Attività')  # Titolo del grafico
plt.axis('equal') # Rendi il grafico a torta circolare

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



3.1.21 Diagramma a Dispersione del Confronto Punteggi di Matematica e Scienze

[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, 82,␣
↪81, 80, 93, 94, 96, 97,85, 92, 78, 88, 95, 90, 89, 86, 79, 91, 84, 87, 83, 82,␣
↪81, 80, 93, 94, 96, 97]

punteggi_scienze = [78, 88, 85, 92, 90, 89, 86, 79, 91, 84, 87, 83, 82, 81, 80,␣
↪93, 94, 96, 97, 75,78, 88, 85, 92, 90, 89, 86, 79, 91, 84, 87, 83, 82, 81, 80,␣
↪93, 94, 96, 97, 75]



# 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')
```

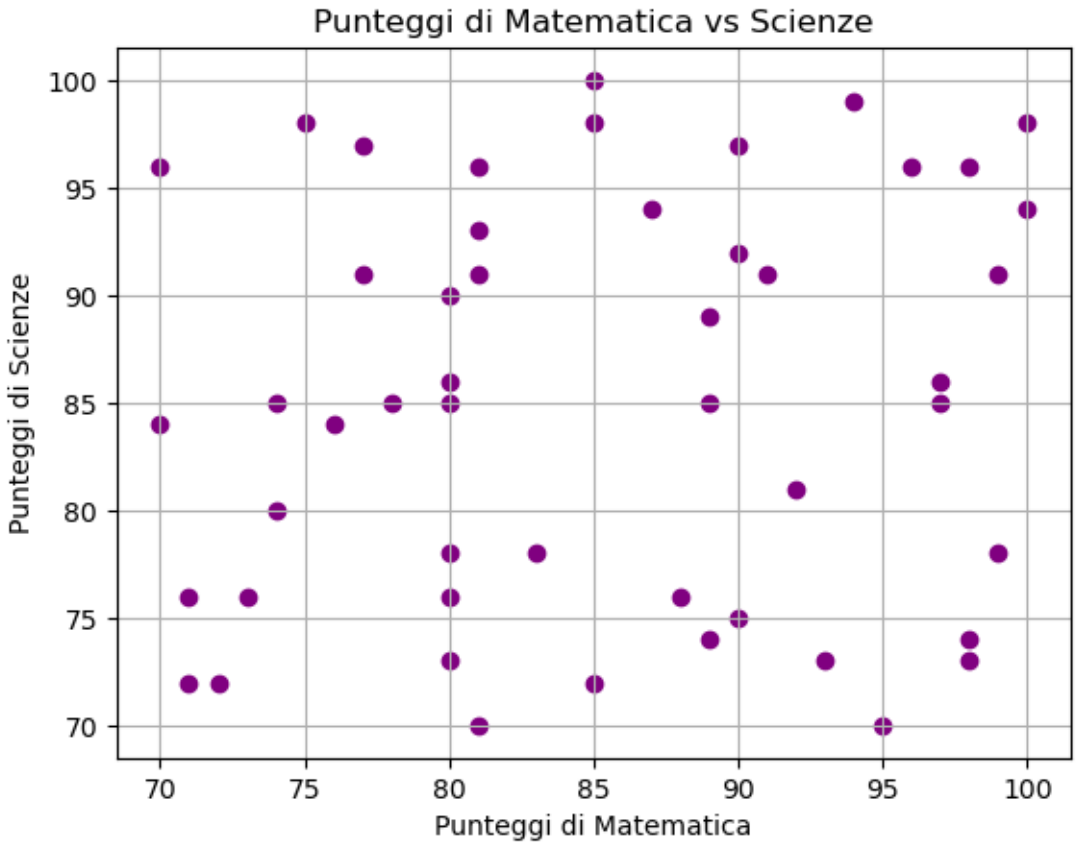
```
plt.ylabel('Punteggi di Scienze')
plt.grid(True)

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



3.1.22 Grafico a Dispersione del Confronto Punteggi di Matematica e Scienze

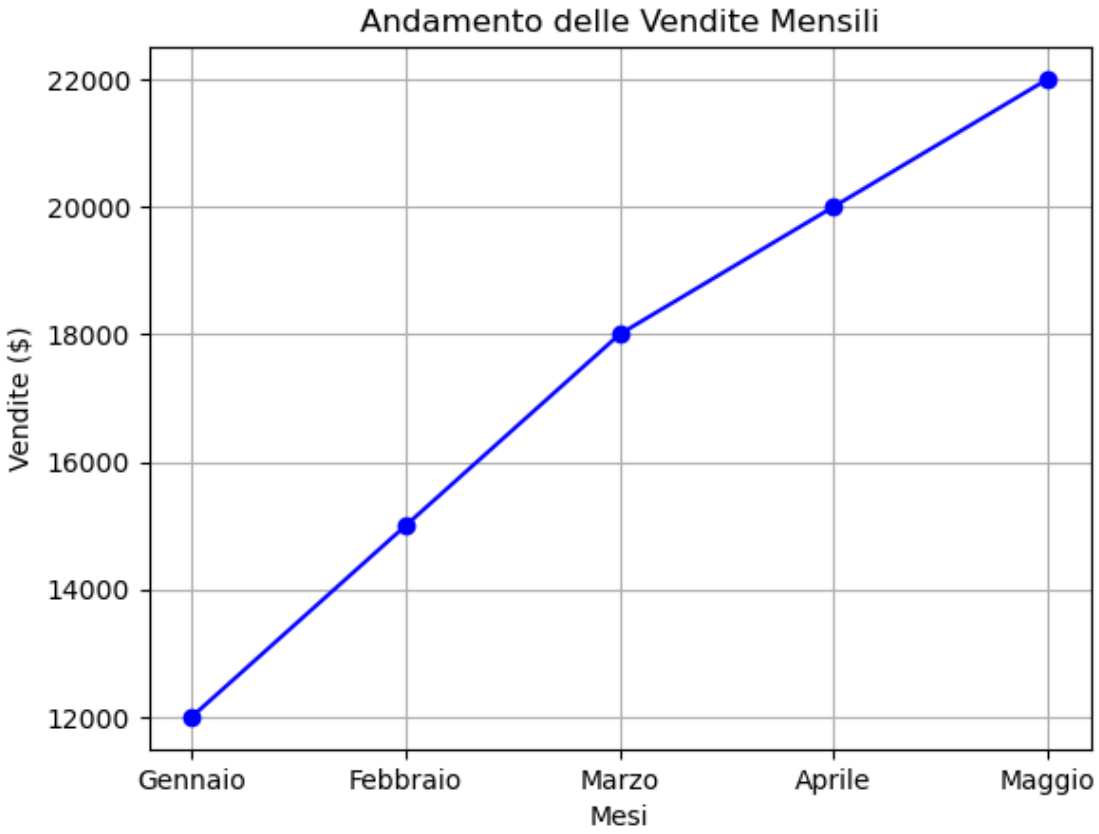
[36]:
```
import random

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()
```



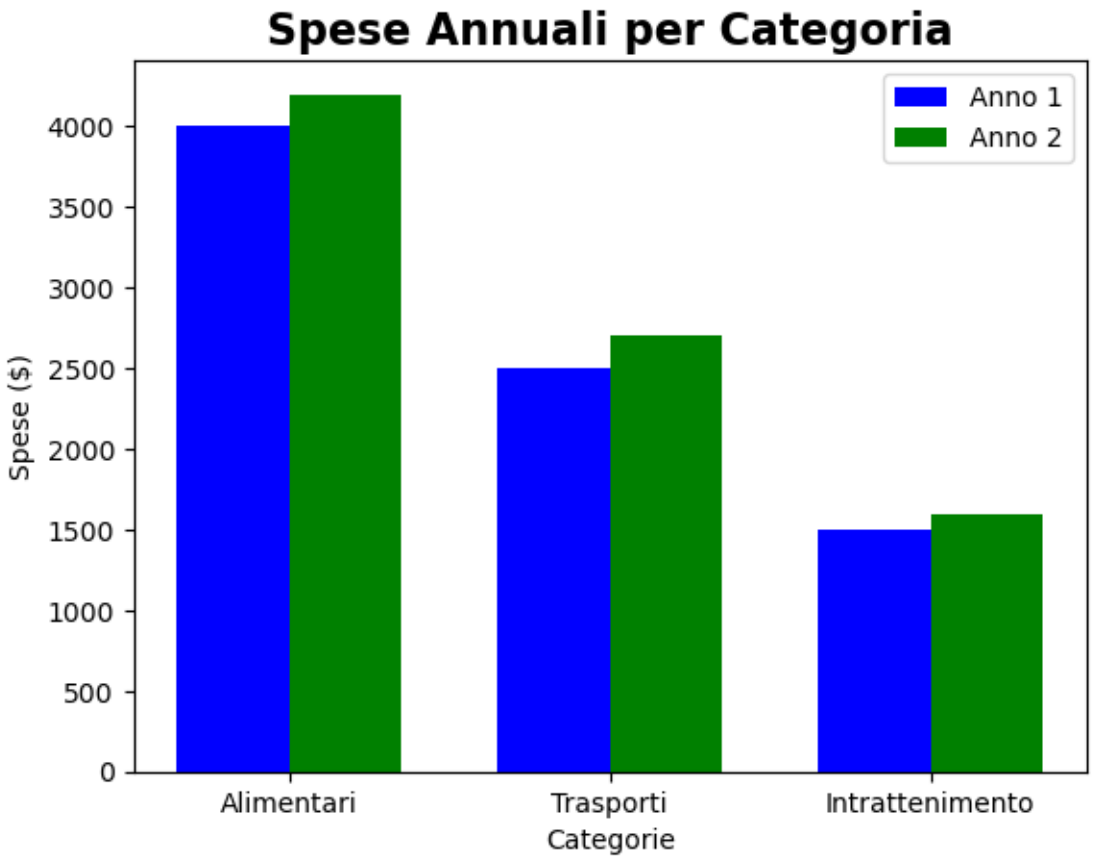
3.1.23 Grafico a Linee del Andamento delle Vendite Mensili

[8]:
```
import matplotlib.pyplot as plt

# Dati di esempio: mesi e corrispondenti vendite
mesi = ['Gennaio', 'Febbraio', 'Marzo', 'Aprile', 'Maggio']
vendite = [12000, 15000, 18000, 20000, 22000]

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

# Personalizza il grafico
plt.title('Andamento delle Vendite Mensili')  # Titolo del grafico
plt.xlabel('Mesi')  # Etichetta dell'asse x
plt.ylabel('Vendite ($)')  # Etichetta dell'asse y
plt.grid(True)  # Mostra la griglia nel grafico

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



3.1.24 Grafico a Barre del Confronto delle Spese Annuali tra Anno 1 e Anno 2

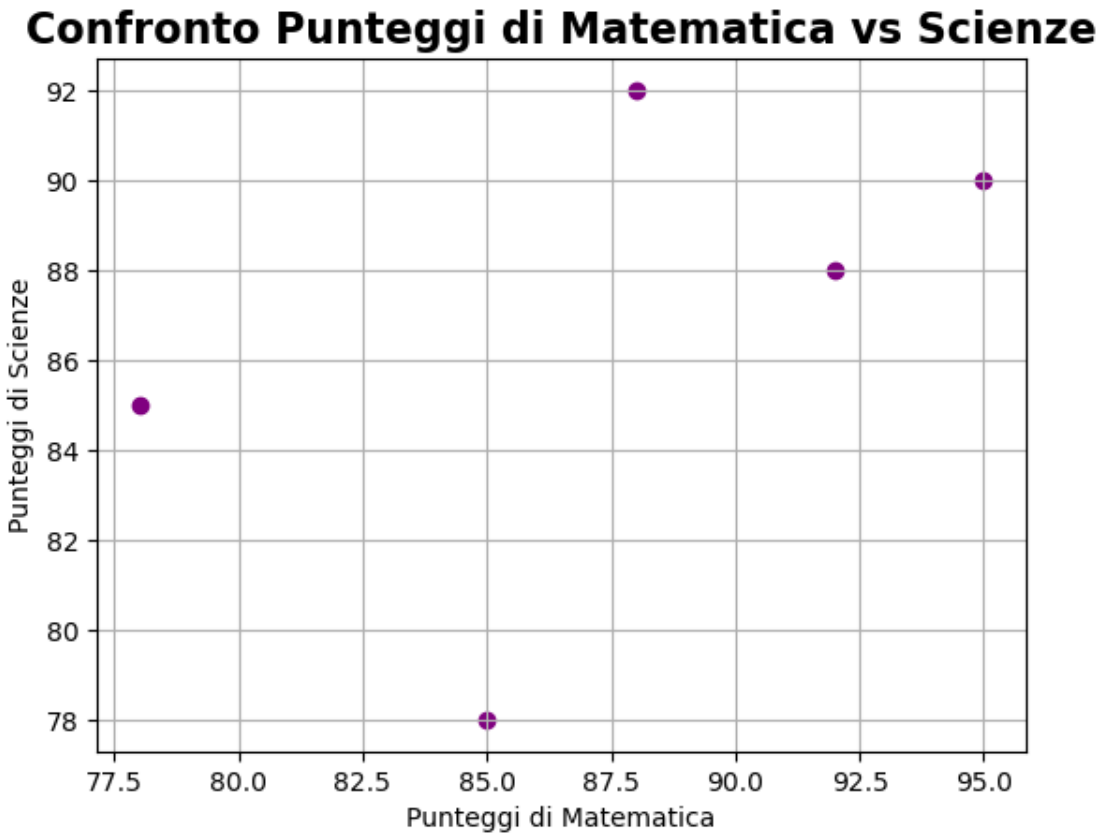
[10]:
```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio: categorie e spese per due anni
categorie = ['Alimentari', 'Trasporti', 'Intrattenimento']
spese_anno_1 = [4000, 2500, 1500]
spese_anno_2 = [4200, 2700, 1600]

# Impostazioni per la larghezza delle barre e gli indici delle categorie
larghezza_barre = 0.35
indici = np.arange(len(categorie))

# Crea un grafico a barre per le spese annuali
plt.bar(indici - larghezza_barre/2, spese_anno_1, width=larghezza_barre,␣
 ↪label='Anno 1', color='blue')
plt.bar(indici + larghezza_barre/2, spese_anno_2, width=larghezza_barre,␣
 ↪label='Anno 2', color='green')

# Personalizza il grafico
plt.title('Spese Annuali per Categoria', fontsize=16, fontweight='bold')  #␣
 ↪Titolo del grafico
plt.xlabel('Categorie')  # Etichetta dell'asse x
plt.ylabel('Spese ($)')  # Etichetta dell'asse y
plt.xticks(indici, categorie)  # Etichette per le categorie sull'asse x
plt.legend(loc='upper right')  # Aggiunta della legenda in alto a destra

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



3.1.25 Grafico a Dispersione del Confronto Punteggi di Matematica e Scienze

[15]:
```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio: punteggi di Matematica e Scienze per un gruppo di studenti
punteggi_matematica = [85, 92, 78, 88, 95]
punteggi_scienze = [78, 88, 85, 92, 90]

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

# Personalizza il grafico
plt.title('Confronto Punteggi di Matematica vs Scienze', fontsize=16,␣
 ↪fontweight='bold')  # Titolo del grafico
plt.xlabel('Punteggi di Matematica')  # Etichetta dell'asse x
plt.ylabel('Punteggi di Scienze')  # Etichetta dell'asse y
plt.grid(True)  # Mostra la griglia nel grafico
```

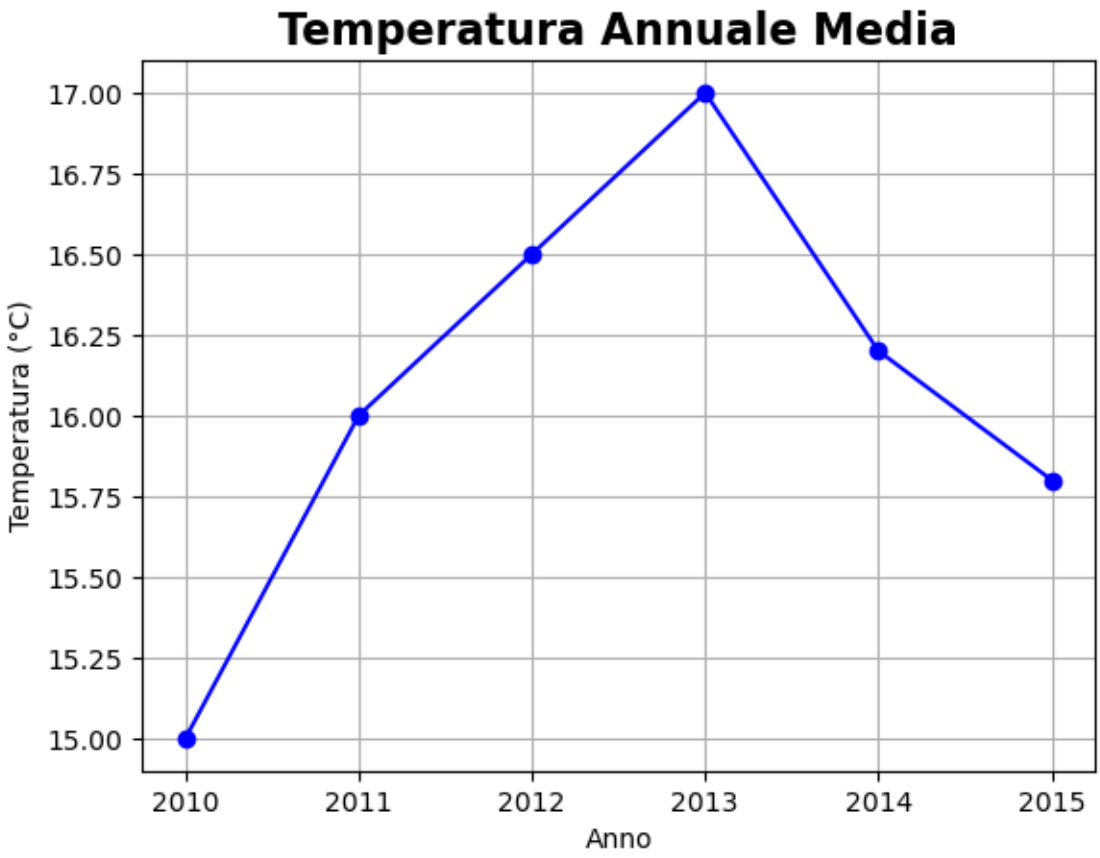
```
# Mostra il grafico risultante
plt.show()
```



3.1.26 Grafico a Linee della Variazione della Temperatura Annuale Media (2010-2015)

[17]:
```
import matplotlib.pyplot as plt

# Dati di esempio: anni e temperatura media annuale
anni = [2010, 2011, 2012, 2013, 2014, 2015]
temperatura_media = [15, 16, 16.5, 17, 16.2, 15.8]

# Crea un grafico a linee
plt.plot(anni, temperatura_media, marker='o', linestyle='-', color='blue')

# Personalizza il grafico
plt.title('Temperatura Annuale Media', fontsize=16, fontweight='bold')  # Titolo␣
 ↪del grafico
plt.xlabel('Anno')  # Etichetta dell'asse x
plt.ylabel('Temperatura (°C)')  # Etichetta dell'asse y
plt.grid(True)  # Mostra la griglia nel grafico
```

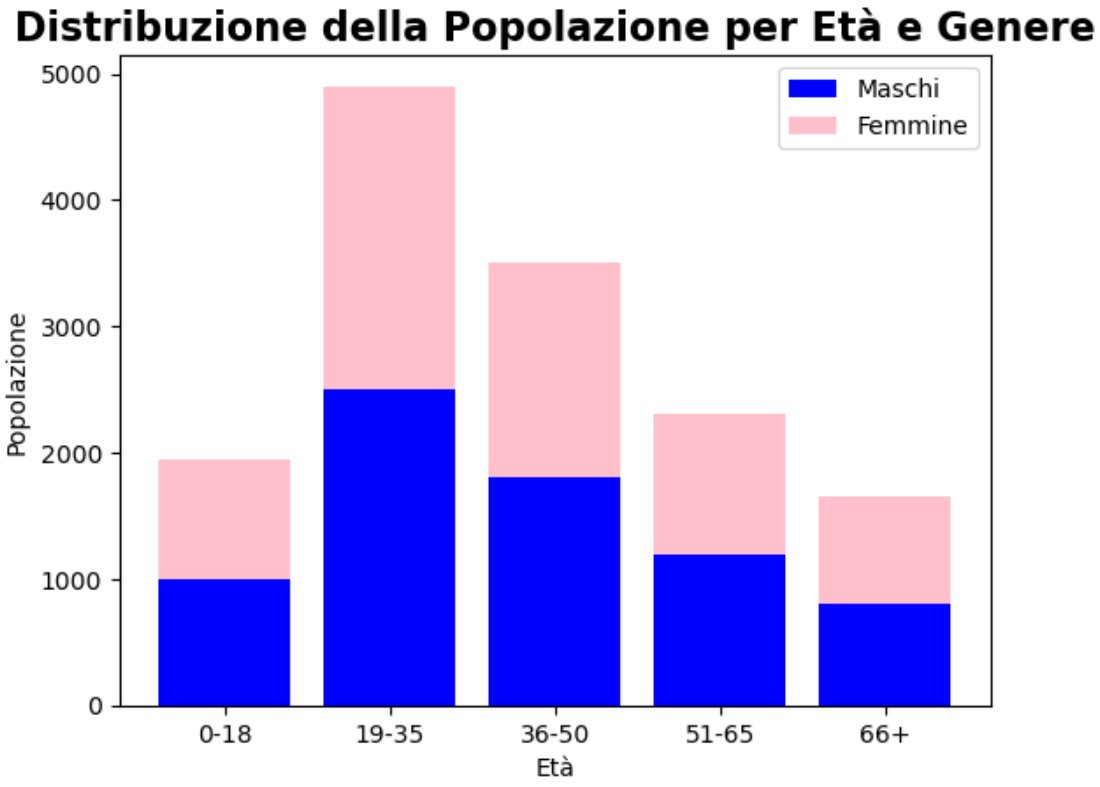
```
# Mostra il grafico risultante
plt.show()
```



3.1.27 Grafico a Barre della Distribuzione della Popolazione per Età e Genere

[18]:
```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio: età, popolazione maschile e popolazione femminile
età = ['0-18', '19-35', '36-50', '51-65', '66+']
popolazione_maschile = [1000, 2500, 1800, 1200, 800]
popolazione_femminile = [950, 2400, 1700, 1100, 850]

# Impostazioni per gli indici delle età
indici = np.arange(len(età))

# Crea un grafico a barre empilate per la distribuzione della popolazione per␣
 ↪età e genere
```

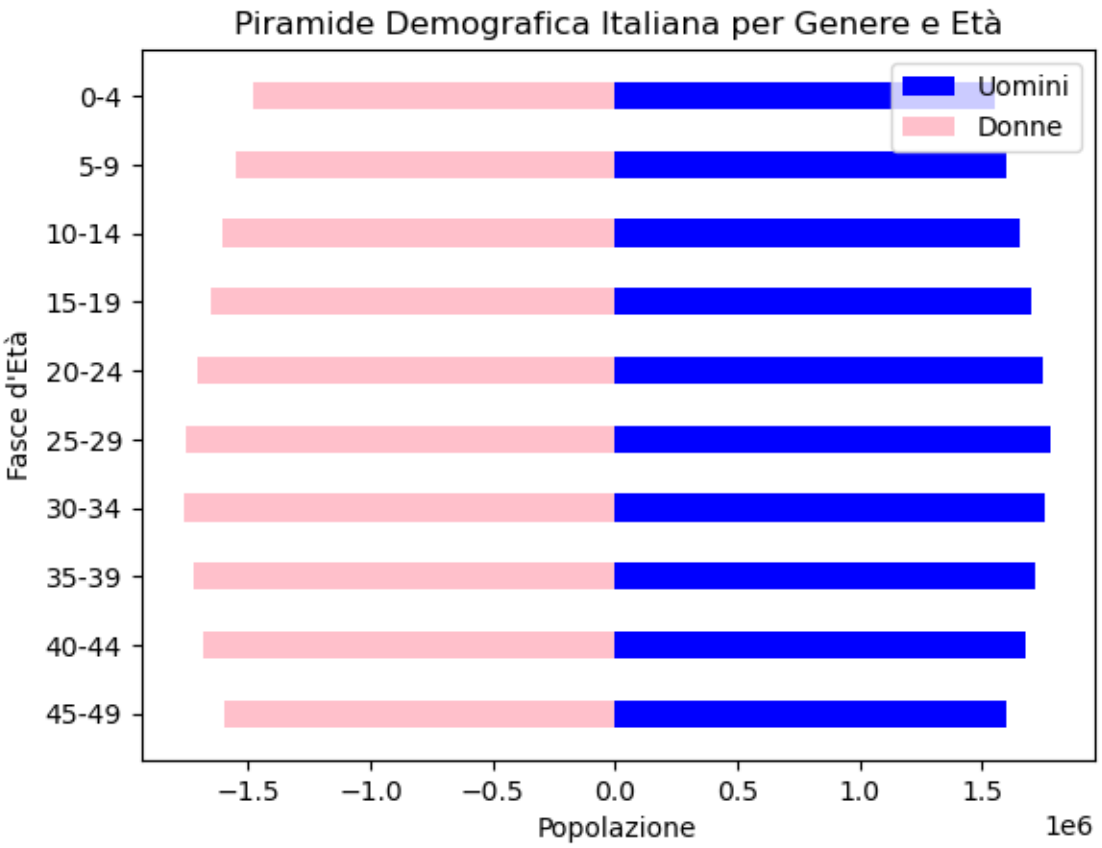
```
plt.bar(età, popolazione_maschile, label='Maschi', color='blue')
plt.bar(età, popolazione_femminile, label='Femmine',␣
 ↪bottom=popolazione_maschile, color='pink')

# Personalizza il grafico
plt.title('Distribuzione della Popolazione per Età e Genere', fontsize=16,␣
 ↪fontweight='bold')  # Titolo del grafico
plt.xlabel('Età')  # Etichetta dell'asse x
plt.ylabel('Popolazione')  # Etichetta dell'asse y
plt.legend(loc='upper right')  # Aggiunta della legenda in alto a destra

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



3.1.28 Piramide Demografica per Genere e per Età

[13]:
```
import matplotlib.pyplot as plt
import numpy as np

# Dati di esempio: fasce d'età, popolazione maschile e popolazione femminile
```

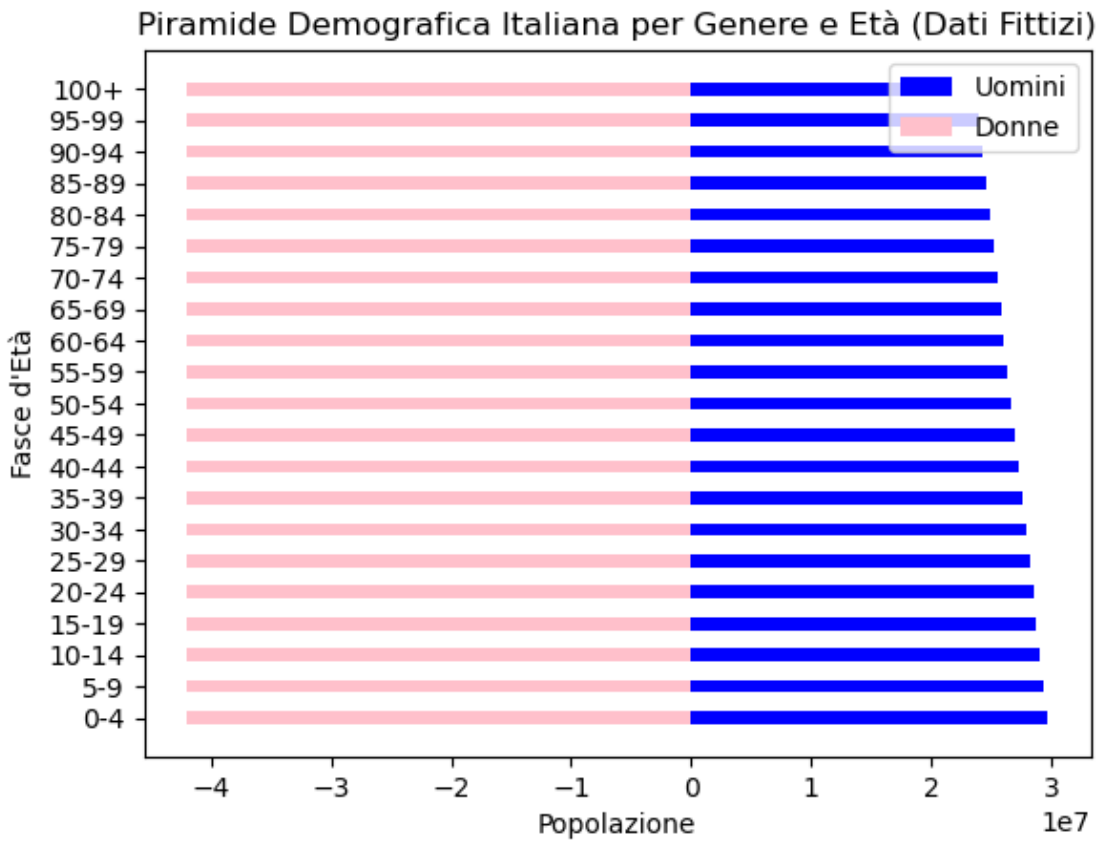
```
fasce_eta = ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29', '30-34', '35-39',␣
 ↪'40-44', '45-49']
popolazione_maschile = [1550000, 1600000, 1650000, 1700000, 1750000, 1780000,␣
 ↪1755000, 1720000, 1675000, 1600000]
popolazione_femminile = [1480000, 1550000, 1605000, 1650000, 1705000, 1750000,␣
 ↪1760000, 1725000, 1680000, 1595000]

# Impostazioni per gli indici delle fasce d'età e la larghezza delle barre
indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

# Creazione della piramide demografica
fig, ax1 = plt.subplots()

ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini',␣
 ↪color='blue')
ax1.barh(indici, np.negative(popolazione_femminile), height=larghezza_barre,␣
 ↪label='Donne', color='pink')

# Personalizzazione del grafico
ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Genere 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
# Mostra il grafico risultante
plt.show()
```



3.1.29 Piramide Demografica Italiana per Genere e Età (Dati Fittizi)

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

# Fasce d'età della popolazione italiana
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', '90-94', '95-99', '100+']

# Popolazione totale italiana (in milioni)
popolazione_totale = 60

# Stima realistica della percentuale di popolazione maschile per ogni fascia di␣
↪età (dati ipotetici)
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, 40.
↪0, 30.0]

popolazione_maschile = []
popolazione_femminile = []

# Calcolo della popolazione maschile per ogni fascia di età
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]

# Impostazioni per gli indici delle fasce d'età e la larghezza delle barre
indici = np.arange(len(fasce_eta))
larghezza_barre = 0.4

# Creazione della piramide demografica
fig, ax1 = plt.subplots()

# Disegna le barre orizzontali per la popolazione maschile (barre blu)
ax1.barh(indici, popolazione_maschile, height=larghezza_barre, label='Uomini',␣
↪color='blue')

# Disegna le barre orizzontali per la popolazione femminile (barre rosa)
ax1.barh(indici, [-x for x in popolazione_femminile], height=larghezza_barre,␣
↪label='Donne', color='pink')

# Personalizzazione del grafico
ax1.set_xlabel('Popolazione')
ax1.set_ylabel('Fasce d\'Età')
ax1.set_title('Piramide Demografica Italiana per Genere e Età (Dati Fittizi)')
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

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



3.1.30 Grafico del Confronto Piramidi Demografiche dei Marchi Auto

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

# Esempio di dati (da sostituire con dati reali)
anni = [2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]
marchi_auto = ['Toyota', 'Ford', 'Chevrolet', 'Honda', 'Nissan', 'Volkswagen',␣
 ↪'Hyundai', 'Mercedes', 'BMW', 'Audi']
vendite_annuali = np.random.randint(50000, 300000, size=(len(marchi_auto),␣
 ↪len(anni)))#size=(len(marchi_auto), len(anni)) indica che stiamo generando una␣
 ↪matrice con il numero di righe pari al numero di marchi auto e il numero di␣
 ↪colonne pari al numero di anni. Questa matrice è quindi utilizzata per␣
 ↪simulare le vendite annuali per ogni marchio auto negli anni specificati.

# Calcola le vendite totali per ogni anno
vendite_totali_annuali = np.sum(vendite_annuali, axis=0)#somma delle vendite per␣
 ↪ogni anno.
```

```
# Creazione di una piramide demografica sovrapposta per ogni marchio
fig, ax = plt.subplots(figsize=(12, 8))#La funzione plt.subplots() in Matplotlib␣
↪viene utilizzata per creare una figura e un set di assi in una singola␣
↪chiamata di funzione. Restituisce un oggetto figura e un array di oggetti assi.

for i, marchio in enumerate(marchi_auto):
    percentuali_per_anno = vendite_annuali[i, :] / vendite_totali_annuali * 100#.
↪ La riga i corrisponde al marchio specifico di interesse, e il simbolo : è uno␣
↪slicing che indica di selezionare tutte le colonne corrispondenti a ciascun␣
↪anno.
    ax.bar(anni, percentuali_per_anno, label=marchio)

ax.set_xlabel('Anno')
ax.set_ylabel('Percentuale di Vendite')
ax.set_title('Confronto Piramidi Demografiche dei Marchi Auto')
ax.legend()
plt.show()
```

4 ESERCITAZIONE 4

4.1 DataFrame

4.1.1 DataFrame con Dati Mancanti

[2]:
```
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
```

[2]:

	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

4.1.2 Selezione delle Colonne “punteggio” e “ammesso” dal DataFrame

[3]:
```
df[["punteggio", "ammesso"]]
```

[3]:
```
   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
```

4.1.3 Identificazione delle righe con dati mancanti

[4]:
```
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
righe_con_dati_mancanti
```

[4]:
```
    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
```

4.1.4 Conta quante righe con dati mancanti ci sono in totale

[5]:
```
totale_dati_mancanti = righe_con_dati_mancanti.shape[0]
totale_dati_mancanti
```

[5]:
```
5
```

4.1.5 Stampa delle Righe con Dati Mancanti e Totale Dati Mancanti dal DataFrame

[6]:
```
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
```

4.1.6 Creazione di un DataFrame con Dati Mancanti in Pandas

[7]:
```
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
```

[7]:

	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

4.1.7 Rimuovi le Righe con Dati Mancanti

[8]:
```
df1=df.dropna(inplace=False)
df1
```

[8]:

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

4.1.8 Esplorazione e Analisi di Dati: Introduzione ai DataFrame con Pandas

[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
```

[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

4.1.9 Trattamento dei missing values nelle variabili numeriche

[10]:
```
numeric_cols = df.select_dtypes(include=['number'])
numeric_cols.columns
```

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

4.1.10 Imputazione dei Valori Mancanti con la Media delle Colonne Numeriche

[11]:
```
df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.
↪columns].mean())
df1
```

[11]:
	Variable1	Variable2
0	1	1.000000
1	2	2.000000
2	3	2.333333
3	4	4.000000
4	5	2.333333

4.1.11 Trattamento dei missing values nelle variabili categoriche

[12]:
```
categorical_cols = df.select_dtypes(exclude=['number'])
categorical_cols.columns
```

[12]:
```
Index(['Missing_Column'], dtype='object')
```

4.1.12 Input dei Valori Mancanti nelle Colonne Categoriali con la Moda

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

[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

4.1.13 Confronto tra Dati Originali e Dati con Input dei Valori Mancanti

[14]:
```
print(f"il primo con i valori mancanti \n{df} \ne il secondo con i missing␣
↪values sostituiti \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
```

4.1.14 Esplorazione e Analisi di Dati: Introduzione al DataFrame con Pandas

[15]:
```
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
```

[15]:

	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

4.1.15 Analisi dei Dati: Conteggio dei Valori Mancanti nelle Feature

[16]:
```
df.isnull().sum()
```

[16]:
```
Feature1    1
Feature2    2
Feature3    1
dtype: int64
```

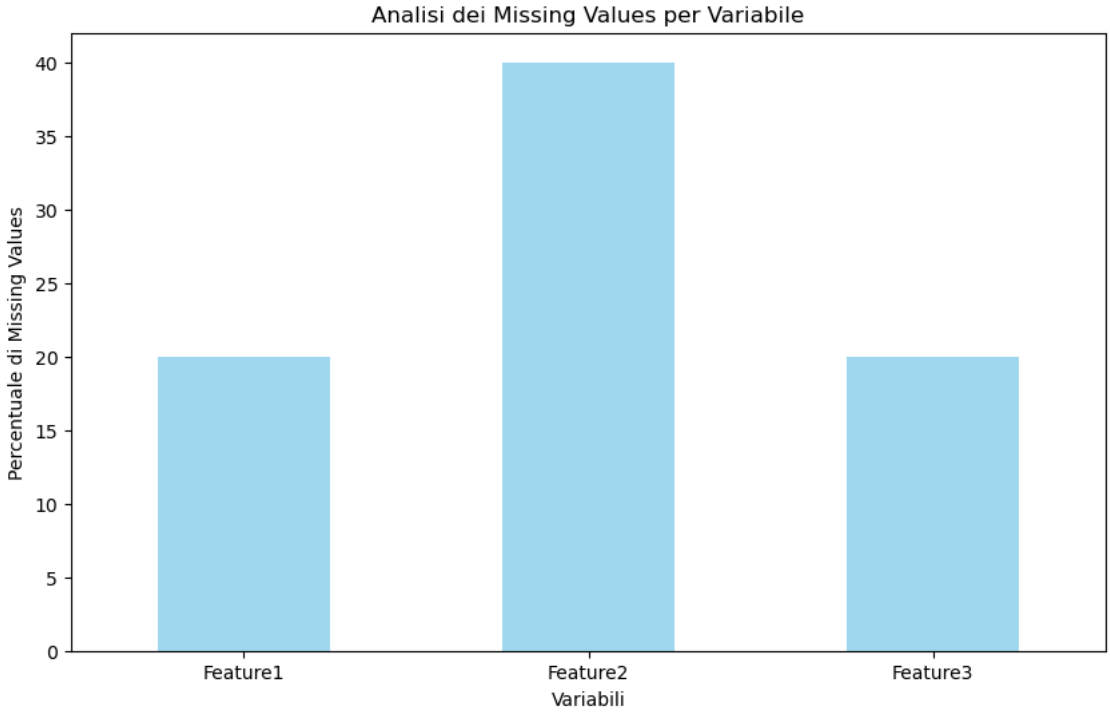
4.1.16 Analisi dei Dati: Percentuale di Valori Mancanti per ciascuna Feature

[17]:
```
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent
```

[17]:
```
Feature1    20.0
Feature2    40.0
Feature3    20.0
dtype: float64
```

4.1.17 Calcola la Percentuale di Righe con Missing Values per Ciascuna Variabile

[18]:
```
# 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()
```



4.1.18 Creo una Funzione che si Opa solo dei Missing Values

[19]:
```
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().iloc[0])
    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
```

4.1.19 Esplorazione e Rilevamento dei Valori Mancanti nel DataFrame

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

[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
```

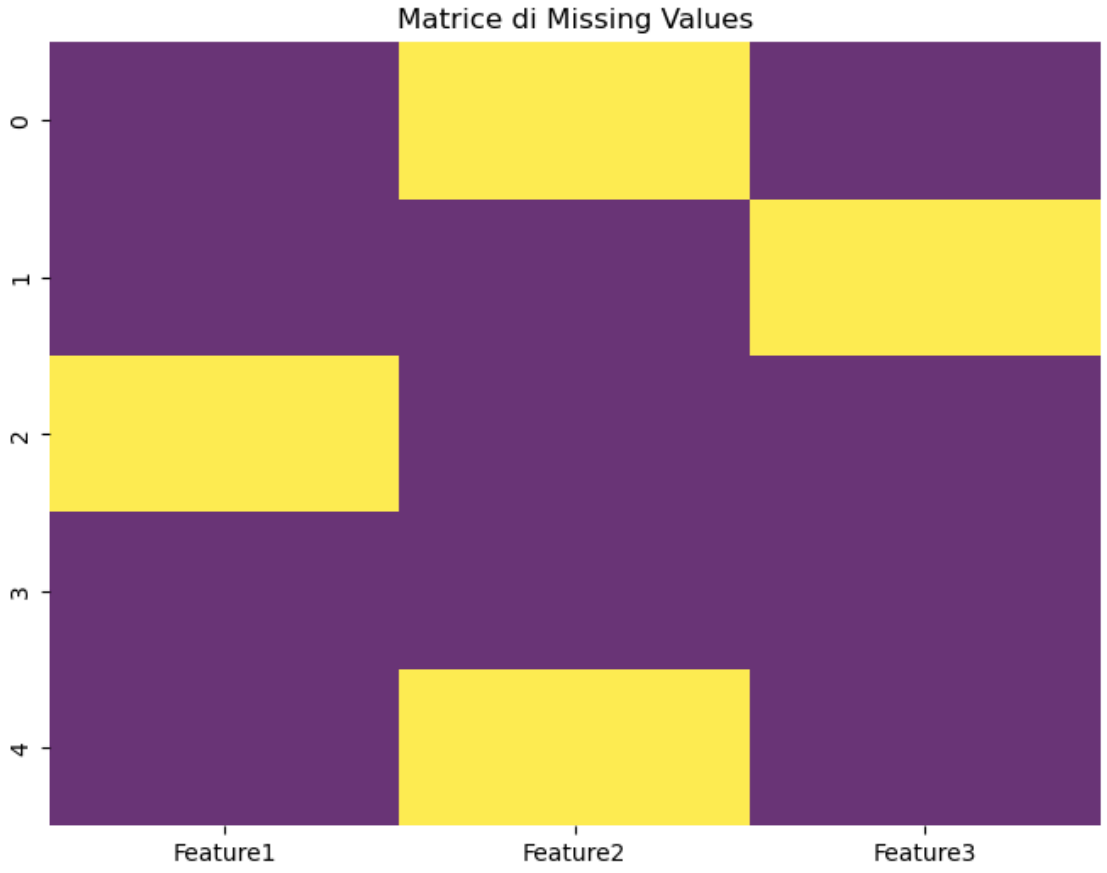
4.1.20 Visualizzazione della Matrice di Valori Mancanti nel DataFrame

[21]:
```
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
```

[21]:
```
   Feature1  Feature2  Feature3
0     False      True     False
1     False     False      True
2      True     False     False
3     False     False     False
4     False      True     False
```

4.1.21 Crea una Heatmap Colorata

[22]:
```
# Visualizzazione della matrice di missing values con un Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(missing_matrix, cmap='viridis', cbar=False, alpha=0.8)
plt.title('Matrice di Missing Values')
plt.show()
```



4.1.22 Esplorazione del

Dataset: Dati Demografici, Punteggi e Reddito

[23]:
```
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
```

4.1.23 Analisi Statistica del Dataset: Informazioni e Statistiche Descrittive

[24]:
```
# 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   int64
 1   Genere     1000 non-null   object
 2   Punteggio  1000 non-null   float64
 3   Reddito    1000 non-null   float64
dtypes: float64(2), int64(1), object(1)
memory usage: 31.4+ 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
```

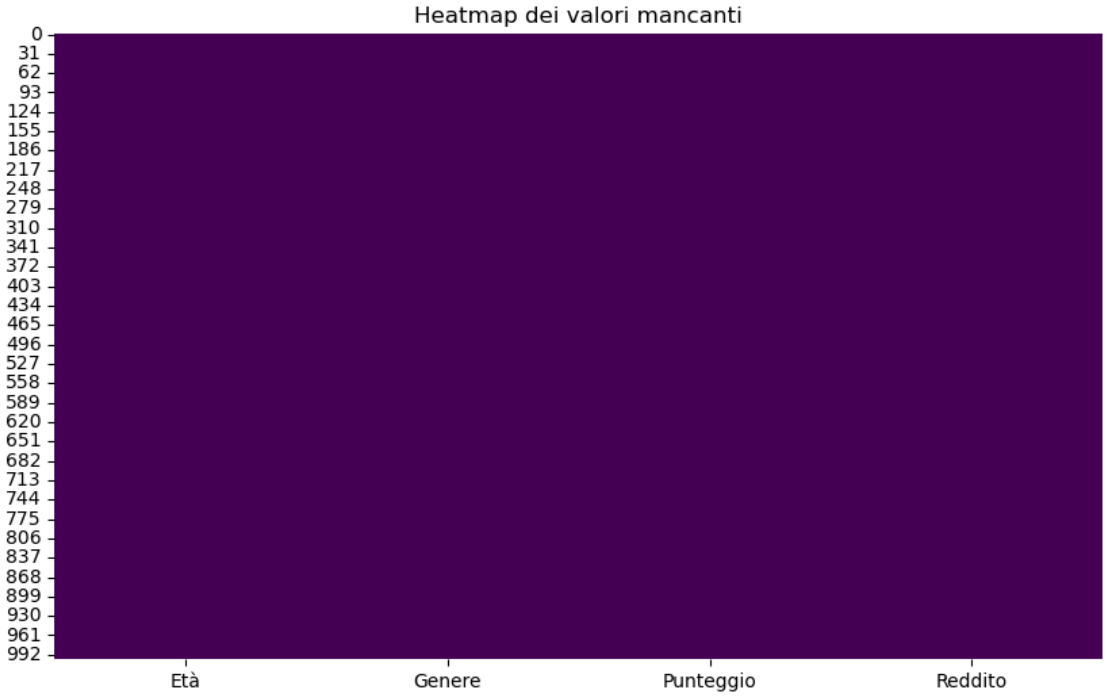
4.1.24 Gestione dei Valori Mancanti nel Dataset: Rilevamento delle Colonne con Dati Mancanti

[25]:
```
# 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
```

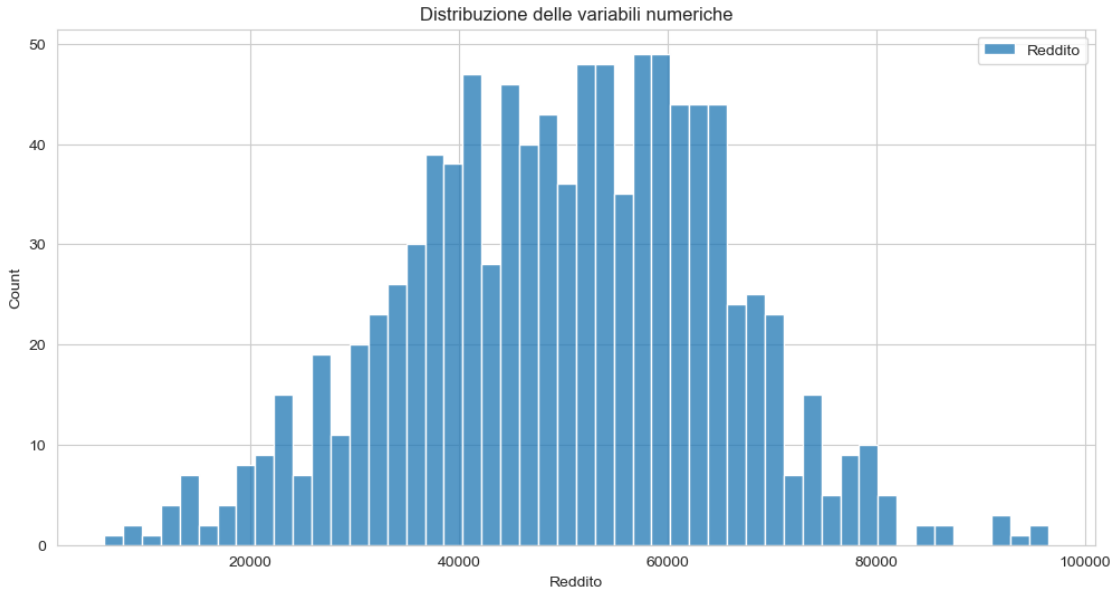
4.1.25 Visualizzazione della Heatmap dei Valori Mancanti nel Dataset

[26]:
```
# Visualizza una heatmap dei valori mancanti
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Heatmap dei valori mancanti')
plt.show()
```



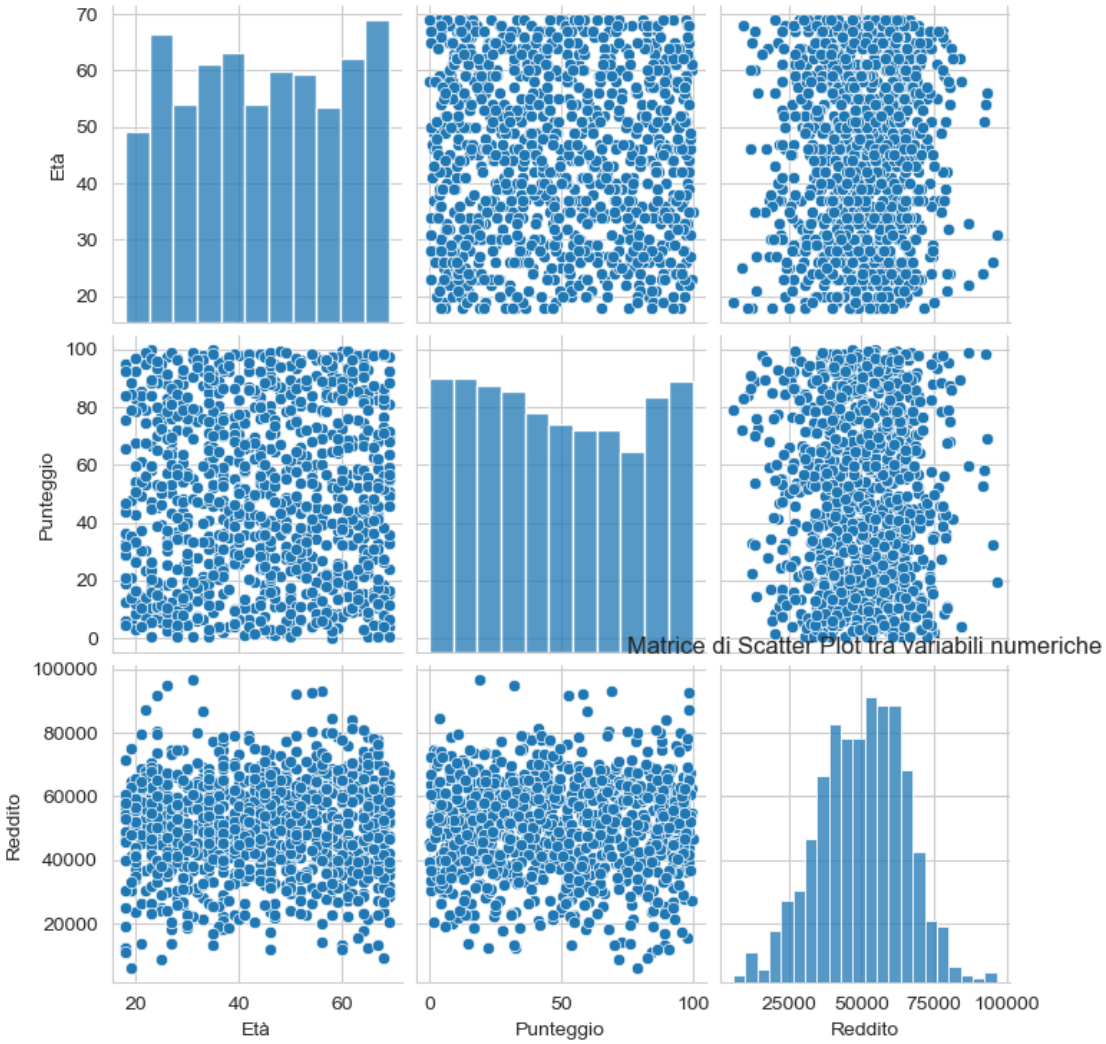
4.1.26 Esplorazione della Distribuzione del Reddito nel Dataset

[27]:
```
# Visualizza la distribuzione delle variabili numeriche
plt.figure(figsize=(12, 6))  # Imposta le dimensioni della figura
sns.set_style("whitegrid")   # Imposta lo stile dello sfondo del grafico
sns.histplot(df["Reddito"], kde=False, bins=50, label="Reddito")  # Crea un␣
↪istogramma del reddito / Bins numero
plt.legend()  # Aggiunge la legenda al grafico
plt.title('Distribuzione delle variabili numeriche')  # Aggiunge il titolo al␣
↪grafico
plt.show()  # Mostra il grafico
```



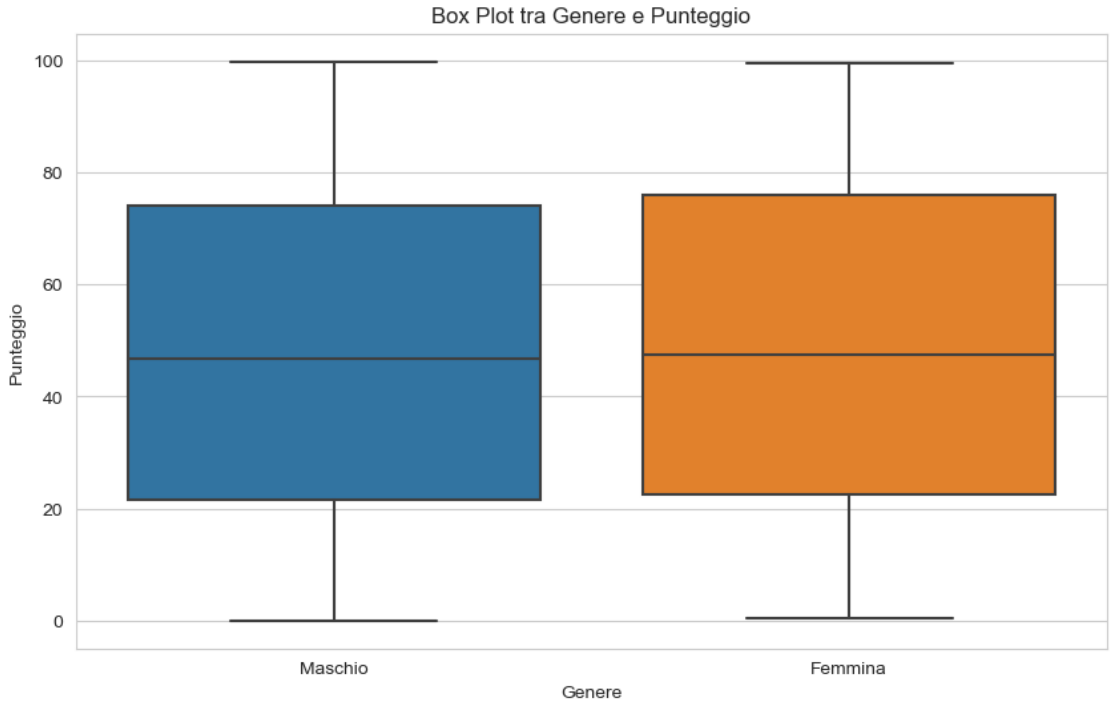
4.1.27 Esplorazione delle Relazioni tra Variabili Numeriche: Matrice di Scatter Plot

[28]:
```
# Visualizza una matrice di scatter plot tra le variabili numeriche
numeric_features = df.select_dtypes(include=[np.number])  # Seleziona solo le␣
↪colonne numeriche
sns.pairplot(df[numeric_features.columns])  # Crea una matrice di scatter plot␣
↪tra le variabili numeriche
plt.title('Matrice di Scatter Plot tra variabili numeriche')  # Aggiunge il␣
↪titolo al grafico
plt.show()  # Mostra il grafico
```



4.1.28 Esplorazione delle Relazioni tra Variabili Categoriche e Numeriche: Box Plot tra Genere e Punteggio

[29]:
```
# Visualizza una box plot per una variabile numerica rispetto a una categorica
plt.figure(figsize=(10, 6))  # Imposta le dimensioni della figura
sns.boxplot(x='Genere', y='Punteggio', data=df)  # Crea un box plot tra Genere e␣
↪Punteggio
plt.title('Box Plot tra Genere e Punteggio')  # Aggiunge il titolo al grafico
plt.show()  # Mostra il grafico
```



4.1.29 Esplorazione delle Relazioni tra Variabili Numeriche e Categoriche: Grafico a Dispersione Interattivo

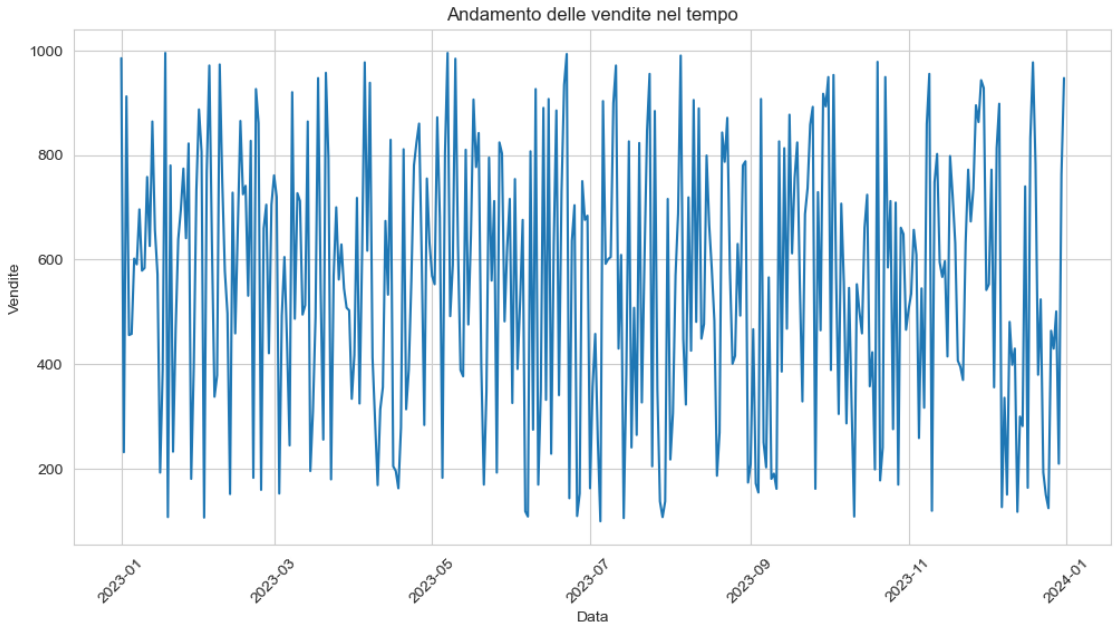
[30]:
```
# Visualizza un grafico a dispersione interattivo utilizzando Plotly
import plotly.express as px  # Importa la libreria Plotly Express

# Crea un grafico a dispersione interattivo
fig = px.scatter(df, x='Età', y='Reddito', color='Genere', size='Punteggio')

# Aggiorna il layout del grafico con un titolo
fig.update_layout(title='Grafico a dispersione interattivo')

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

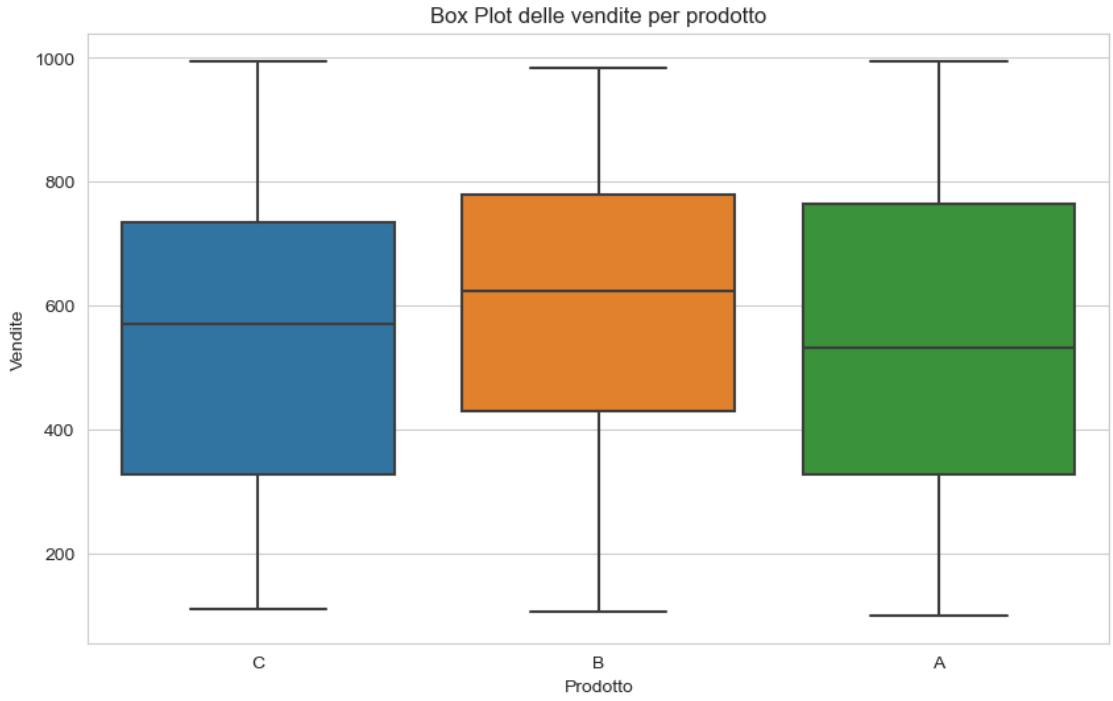
4.1.30 Esplorazione delle Vendite: Analisi Temporale e per Prodotto

[31]:
```
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)  # Imposta il seed per la riproducibilità dei dati casuali
```

```
data = {
    'Data': pd.date_range(start='2023-01-01', end='2023-12-31', freq='D'),  #␣
 ↪Crea una serie di date giornaliere per l'intero anno 2023
    'Vendite': np.random.randint(100, 1000, size=365),  # Genera dati casuali␣
 ↪per le vendite giornaliere
    'Prodotto': np.random.choice(['A', 'B', 'C'], size=365)  # Genera dati␣
 ↪casuali per il tipo di prodotto venduto
}

df = pd.DataFrame(data)  # Crea un DataFrame con i dati generati

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

# Visualizza un grafico delle vendite nel tempo
plt.figure(figsize=(12, 6))  # Imposta le dimensioni della figura
sns.lineplot(x='Data', y='Vendite', data=df)  # Crea un grafico a linea delle␣
 ↪vendite nel tempo
plt.title('Andamento delle vendite nel tempo')  # Aggiunge il titolo al grafico
plt.xlabel('Data')  # Etichetta l'asse x come "Data"
plt.ylabel('Vendite')  # Etichetta l'asse y come "Vendite"
plt.xticks(rotation=45)  # Ruota le etichette sull'asse x di 45 gradi per una␣
 ↪migliore leggibilità
plt.show()  # Mostra il grafico

# Visualizza una box plot delle vendite per prodotto
plt.figure(figsize=(10, 6))  # Imposta le dimensioni della nuova figura
sns.boxplot(x='Prodotto', y='Vendite', data=df)  # Crea un box plot delle␣
 ↪vendite per prodotto
plt.title('Box Plot delle vendite per prodotto')  # Aggiunge il titolo al grafico
plt.xlabel('Prodotto')  # Etichetta l'asse x come "Prodotto"
plt.ylabel('Vendite')  # Etichetta l'asse y come "Vendite"
plt.show()  # Mostra il grafico
```

```
        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
```


Andamento delle vendite nel tempo
Vendite
1000
800
600
400
200
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01
Data


Box Plot delle vendite per prodotto
Vendite
1000
800
600
400
200
C
B
A
Prodotto

4.1.31 Esplorazione di Dati: DataFrame con Variabili Numeriche e Categoriche

[32]:
```
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],  # Dati numerici, includendo un␣
 ↪valore mancante
    'Categorical_Var': ['A', 'B', 'A', 'B', 'A', 'B']  # Dati categorici
}

# Crea un DataFrame
df = pd.DataFrame(data)  # Crea un DataFrame utilizzando i dati forniti
print(df)  # Stampa il DataFrame per l'ispezione
```

```
   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
```

4.1.32 Calcolo della Media Condizionata sulla Variabile Numerica

[33]:
```
# Calcola la media condizionata
conditional_means = df['Numeric_Var'].fillna(df.
 ↪groupby('Categorical_Var')['Numeric_Var'].transform('mean'))
```

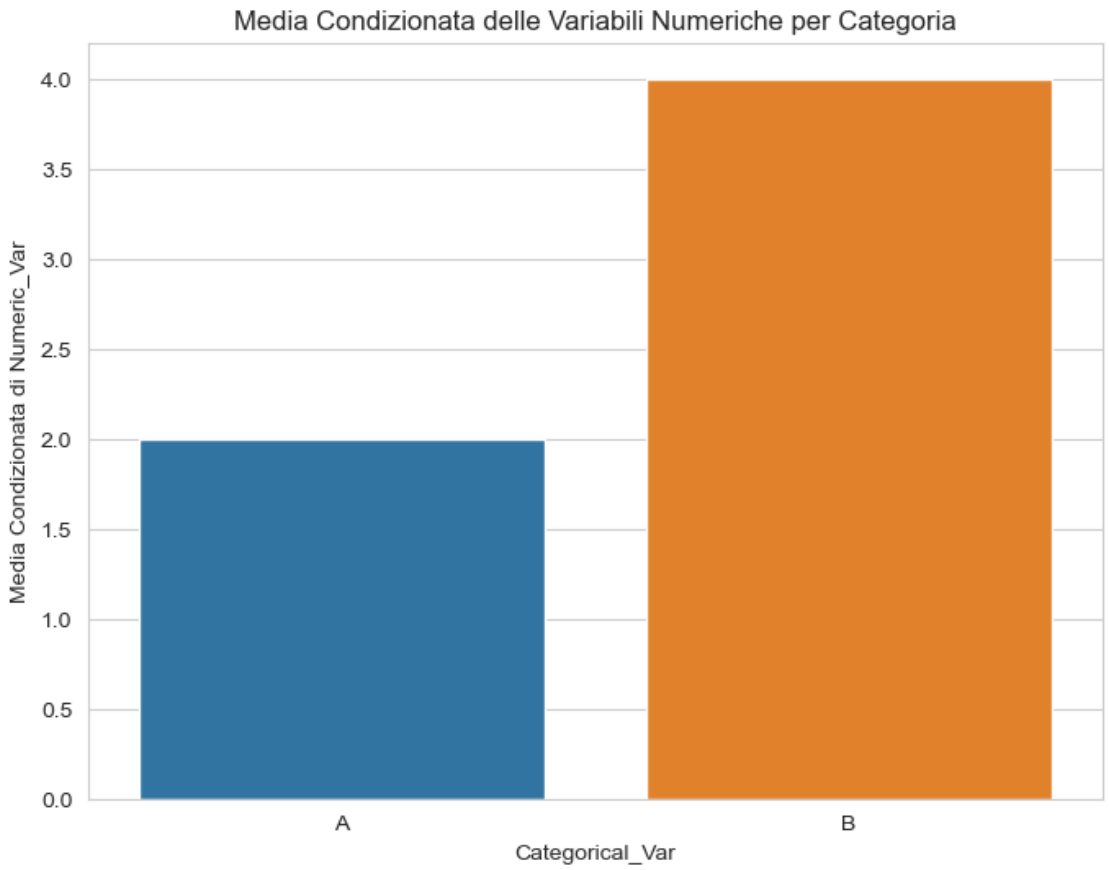
4.1.33 Aggiornamento della Colonna Numeric_Var con la Media Condizionata e Grafico a Barre della Media Condizionata per ogni Categoria

[37]:
```
# 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))  # Imposta le dimensioni della figura
sns.barplot(data=df, x='Categorical_Var', y='Numeric_Var', ci=False)  # Crea un␣
 ↪grafico a barre utilizzando Seaborn
plt.xlabel('Categorical_Var')  # Etichetta l'asse x come "Categorical_Var"
plt.ylabel('Media Condizionata di Numeric_Var')  # Etichetta l'asse y come␣
 ↪"Media Condizionata di Numeric_Var"
plt.title('Media Condizionata delle Variabili Numeriche per Categoria')  #␣
 ↪Aggiunge il titolo al grafico
```

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

```
   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
```

```
/var/folders/3b/sxxjhxkj11v71719m2y57znm0000gn/T/ipykernel_2041/680230299.py:7:
FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same
effect.
```

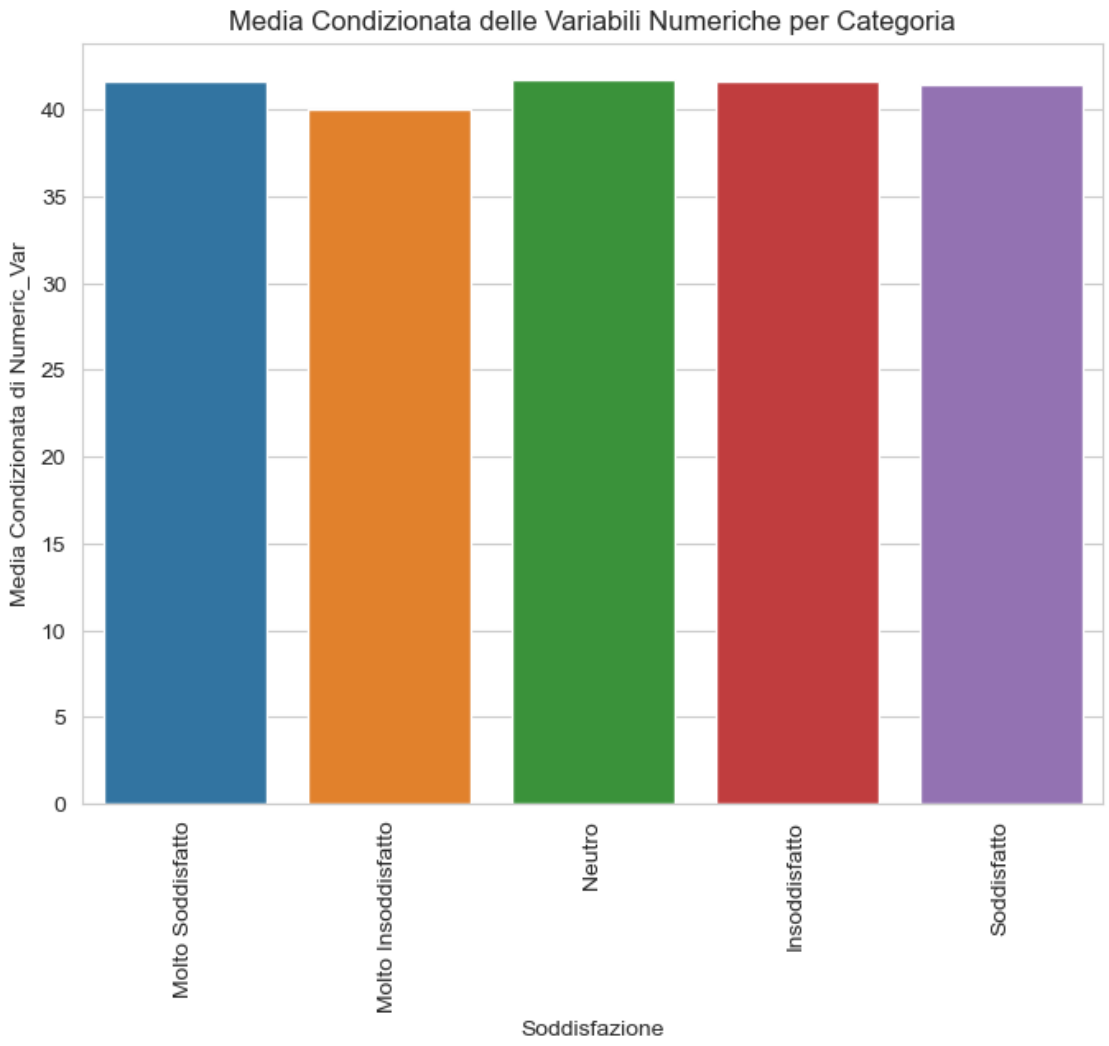


4.1.34 Esplorazione della Soddisfazione e Media Condizionata dell'Età

[38]:
```
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)  # Imposta il seed per la riproducibilità dei dati casuali
data = {
    'Età': np.random.randint(18, 65, size=500),  # Dati casuali per l'età, compresi tra 18 e 65 anni
    'Soddisfazione': np.random.choice(['Molto Soddisfatto', 'Soddisfatto', 'Neutro', 'Insoddisfatto', 'Molto Insoddisfatto'], size=500)  # Dati casuali per la soddisfazione
}

df = pd.DataFrame(data)  # Crea un DataFrame con i dati generati
print(df)  # Stampa il DataFrame per l'ispezione

# Calcola la media condizionata sulla variabile 'Età' rispetto a 'Soddisfazione'
conditional_means = df.groupby('Soddisfazione')['Età'].transform('mean')

# Aggiunge una nuova colonna 'Numeric_Var' al DataFrame con la media condizionata
df['Numeric_Var'] = conditional_means
print(df)  # Stampa il DataFrame aggiornato per l'ispezione

# Crea un grafico a barre per mostrare la media condizionata per ogni categoria di soddisfazione
plt.figure(figsize=(8, 6))  # Imposta le dimensioni della figura
sns.barplot(data=df, x='Soddisfazione', y='Numeric_Var', ci=None)  # Crea un grafico a barre utilizzando Seaborn
plt.xlabel('Soddisfazione')  # Etichetta l'asse x come "Soddisfazione"
plt.ylabel('Media Condizionata di Numeric_Var')  # Etichetta l'asse y come "Media Condizionata di Numeric_Var"
plt.title('Media Condizionata delle Variabili Numeriche per Categoria')  # Aggiunge il titolo al grafico
plt.xticks(rotation=90)  # Ruota le etichette sull'asse x di 90 gradi per una migliore leggibilità
plt.show()  # Mostra il grafico
```

```
     Età        Soddisfazione
0     56    Molto Soddisfatto
1     46  Molto Insoddisfatto
2     32              Neutro
3     60              Neutro
4     25  Molto Insoddisfatto
..   ...                 ...
```

```
495   37    Molto Soddisfatto
496   41    Molto Soddisfatto
497   29    Molto Soddisfatto
498   52    Molto Soddisfatto
499   50    Molto Soddisfatto

[500 rows x 2 columns]
     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]

/var/folders/3b/sxxjhxkj11v71719m2y57znm0000gn/T/ipykernel_2041/2963085462.py:25
: FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

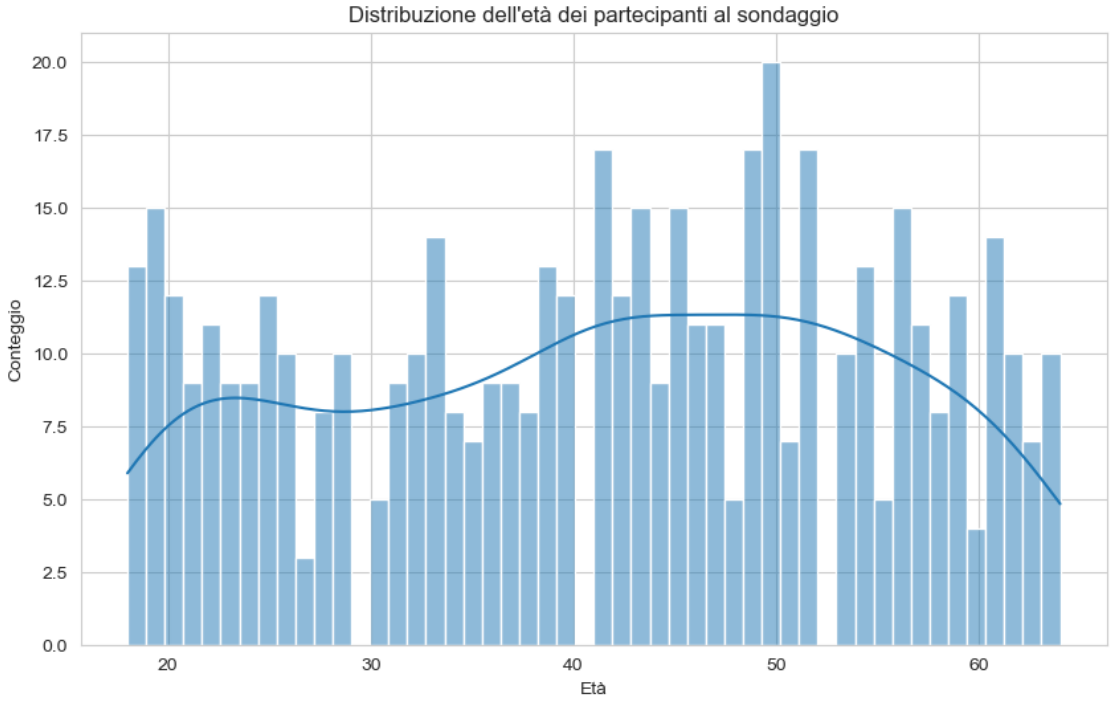


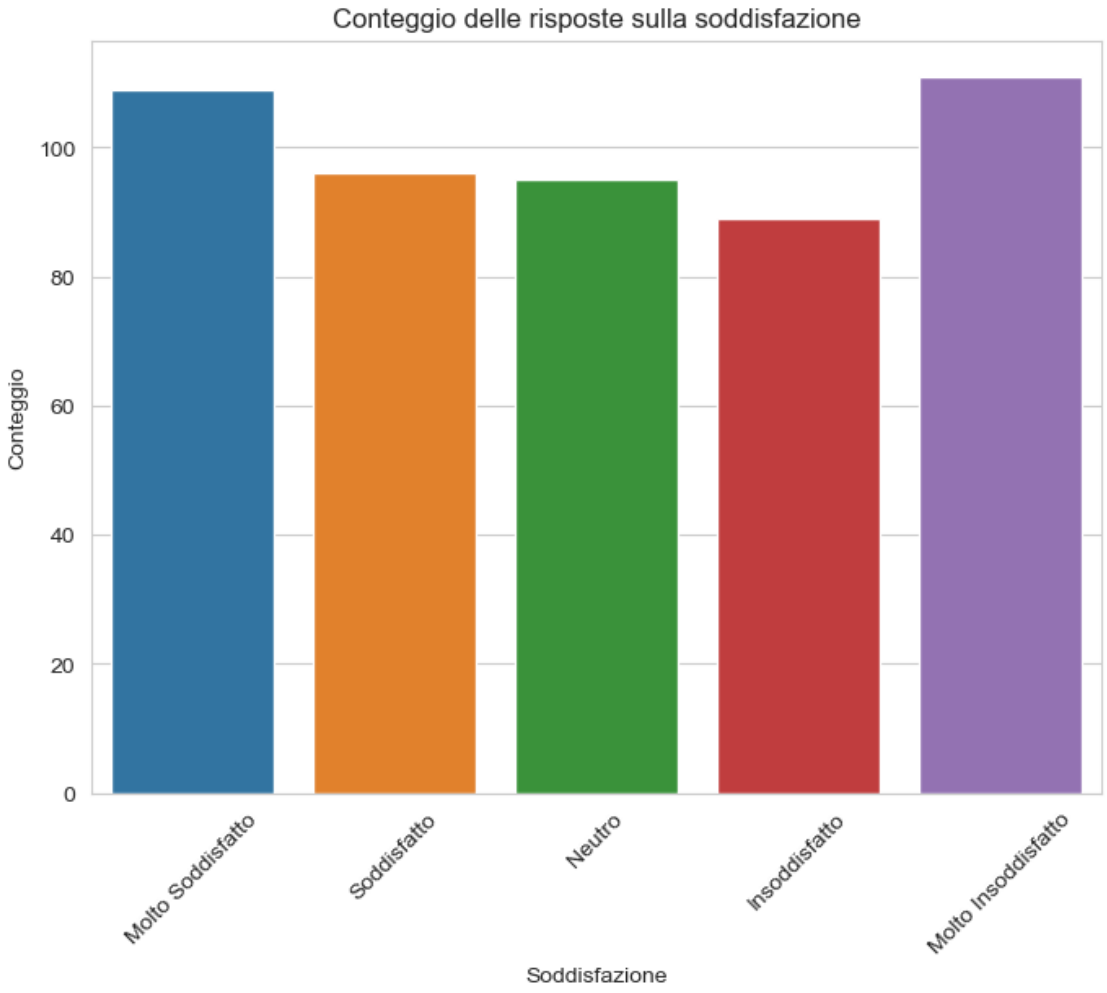
4.1.35 "Esplorazione del Dataset: Prima Righe, Distribuzione dell'Età e Conteggio delle Risposte sulla Soddisfazione"

[39]:
```
# Visualizza le prime righe del dataset
print(df.head())

# Visualizza una distribuzione dell'età
plt.figure(figsize=(10, 6))  # Imposta le dimensioni della figura
sns.histplot(df['Età'], bins=50, kde=True)  # Crea un istogramma dell'età con una sovrapposizione della stima della densità kernel (kde)
plt.title('Distribuzione dell\'età dei partecipanti al sondaggio')  # Aggiunge il titolo al grafico
plt.xlabel('Età')  # Etichetta l'asse x come "Età"
plt.ylabel('Conteggio')  # Etichetta l'asse y come "Conteggio"
```

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

# Visualizza un conteggio delle risposte sulla soddisfazione
plt.figure(figsize=(8, 6))  # Imposta le dimensioni della nuova figura
sns.countplot(x='Soddisfazione', data=df, order=['Molto Soddisfatto',␣
 ↪'Soddisfatto', 'Neutro', 'Insoddisfatto', 'Molto Insoddisfatto'])  # Crea un␣
 ↪grafico a barre del conteggio delle risposte sulla soddisfazione
plt.title('Conteggio delle risposte sulla soddisfazione')  # Aggiunge il titolo␣
 ↪al grafico
plt.xlabel('Soddisfazione')  # Etichetta l'asse x come "Soddisfazione"
plt.ylabel('Conteggio')  # Etichetta l'asse y come "Conteggio"
plt.xticks(rotation=45)  # Ruota le etichette sull'asse x di 45 gradi per una␣
 ↪migliore leggibilità
plt.show()  # Mostra il grafico
```

	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





4.1.36 Generazione di una Matrice 100x5 con Valori Casuali compresi tra 0 (incluso) e 1 (escluso)

[40]:
```
# Genera un array bidimensionale (matrice) di dimensioni 100x5 con valori casuali
np.random.rand(100, 5)
```

[40]:
```
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,
 7.26095586e-01],
[5.92962905e-01, 1.02212664e-01, 9.18750522e-01, 7.90084637e-01,
 2.30226048e-02],
[6.51367211e-01, 7.71246669e-01, 3.74435370e-01, 6.89215394e-02,
 7.73210655e-02],
[1.04246966e-01, 8.40439632e-01, 9.10686198e-01, 1.22810782e-01,
 2.35906035e-01],
[1.65520510e-01, 1.86320879e-01, 8.37490789e-01, 3.32146413e-01,
 3.11444264e-01],
[2.27395621e-01, 6.07894394e-01, 3.79305698e-01, 7.44249652e-01,
 2.05579808e-01],
[7.87789889e-01, 6.03721532e-01, 1.14267300e-01, 4.14504584e-01,
 8.63518141e-01],
[9.22960232e-01, 4.65700122e-01, 4.80836978e-01, 9.18454725e-01,
 5.87068224e-01],
[3.28466702e-02, 9.12749463e-01, 2.48226443e-01, 5.77631954e-01,
 1.65513317e-01],
[3.38791602e-02, 3.11473171e-01, 7.80495708e-01, 2.77587173e-01,
 2.20097214e-01],
[2.12680784e-01, 5.15173682e-01, 9.75541316e-01, 4.58988988e-01,
 5.57305369e-01],
[8.60632018e-01, 5.35069719e-01, 1.84462871e-01, 2.99595912e-01,
 3.09936926e-01],
[3.97318115e-01, 4.26788905e-01, 7.99782022e-01, 3.49386999e-01,
 4.68231828e-01],
[6.24975987e-01, 3.77725947e-01, 8.36564813e-01, 5.87453604e-01,
 2.94044679e-01],
[7.14059200e-01, 5.27647730e-01, 5.34455891e-01, 4.81087948e-01,
 4.97011949e-01],
[7.65356281e-01, 1.02981479e-01, 3.34362811e-01, 7.55033638e-02,
 7.53246221e-01],
[2.72301855e-01, 8.97430410e-01, 5.26578467e-01, 8.00749771e-01,
 9.78931436e-01],
[8.39788859e-01, 8.66993760e-01, 4.07984495e-01, 5.51722595e-01,
 2.53888564e-01],
[1.96113100e-01, 5.05507879e-01, 5.95048980e-01, 3.39271246e-01,
 5.69443518e-01],
[8.87460483e-01, 5.56721585e-01, 7.20813096e-01, 8.05314967e-01,
```

```
 9.88728782e-01],
[6.03197402e-01, 8.06850149e-01, 9.62649319e-01, 9.44409726e-01,
 1.41081849e-01],
[4.06392308e-01, 3.23990061e-01, 8.69250150e-02, 6.33240775e-01,
 7.35904613e-01],
[8.48128095e-01, 1.22776890e-01, 8.76444076e-01, 6.42925564e-01,
 7.03946135e-01],
[9.10624004e-01, 6.24761271e-01, 3.35865857e-01, 8.25107331e-01,
 3.63059212e-01],
[3.42281108e-02, 8.30654507e-01, 3.45192145e-01, 7.73834523e-01,
 3.62757895e-01],
[8.61068020e-01, 2.19510525e-01, 9.74547265e-01, 7.79758775e-01,
 1.14237926e-01],
[5.65776620e-01, 9.85367312e-01, 4.71065453e-01, 1.82107139e-01,
 4.84778367e-01],
[5.12457656e-01, 7.41686944e-01, 6.98845853e-01, 4.02550044e-01,
 2.18023107e-01],
[6.45054168e-01, 4.21703935e-01, 1.32276211e-01, 9.00572301e-01,
 6.91221858e-01],
[6.83183856e-01, 8.22583876e-01, 5.29661679e-01, 8.15054428e-01,
 4.98090754e-01],
[6.71753999e-02, 4.05248088e-01, 4.97464894e-01, 7.20202582e-01,
 1.06578730e-01],
[1.40018539e-01, 2.63683803e-01, 2.66498393e-01, 7.32184578e-01,
 2.50730661e-01],
[6.33098063e-01, 4.94273850e-01, 5.73294378e-01, 8.38106963e-01,
 4.04381502e-01],
[7.70911377e-01, 4.22036060e-01, 3.53871377e-01, 9.58627177e-01,
 1.84370921e-01],
[9.03369606e-02, 7.70985926e-01, 6.82497054e-02, 8.37078991e-01,
 4.37193396e-01],
[9.15513677e-01, 7.21532861e-01, 6.10404667e-01, 9.49306797e-01,
 3.96649373e-01],
[9.54086937e-01, 1.35001277e-01, 4.83754942e-01, 2.69911222e-01,
 5.39842779e-01],
[1.62013878e-01, 8.41529660e-01, 8.46407849e-01, 9.55236313e-01,
 1.54672141e-01],
[6.23332551e-01, 4.74237309e-01, 3.44383732e-01, 3.51540161e-01,
 4.12793551e-01],
[7.04925550e-01, 5.98286058e-01, 4.59681513e-01, 7.59334104e-02,
 7.77653441e-02],
[2.33939063e-03, 9.67699388e-01, 5.25381856e-03, 1.03093746e-01,
 3.14596557e-01],
[8.07430252e-01, 9.62098786e-01, 7.90013559e-01, 6.90560232e-01,
 5.21300667e-01],
[8.67052119e-02, 9.58170793e-01, 7.58392414e-01, 6.38949385e-01,
 7.59035846e-01],
```

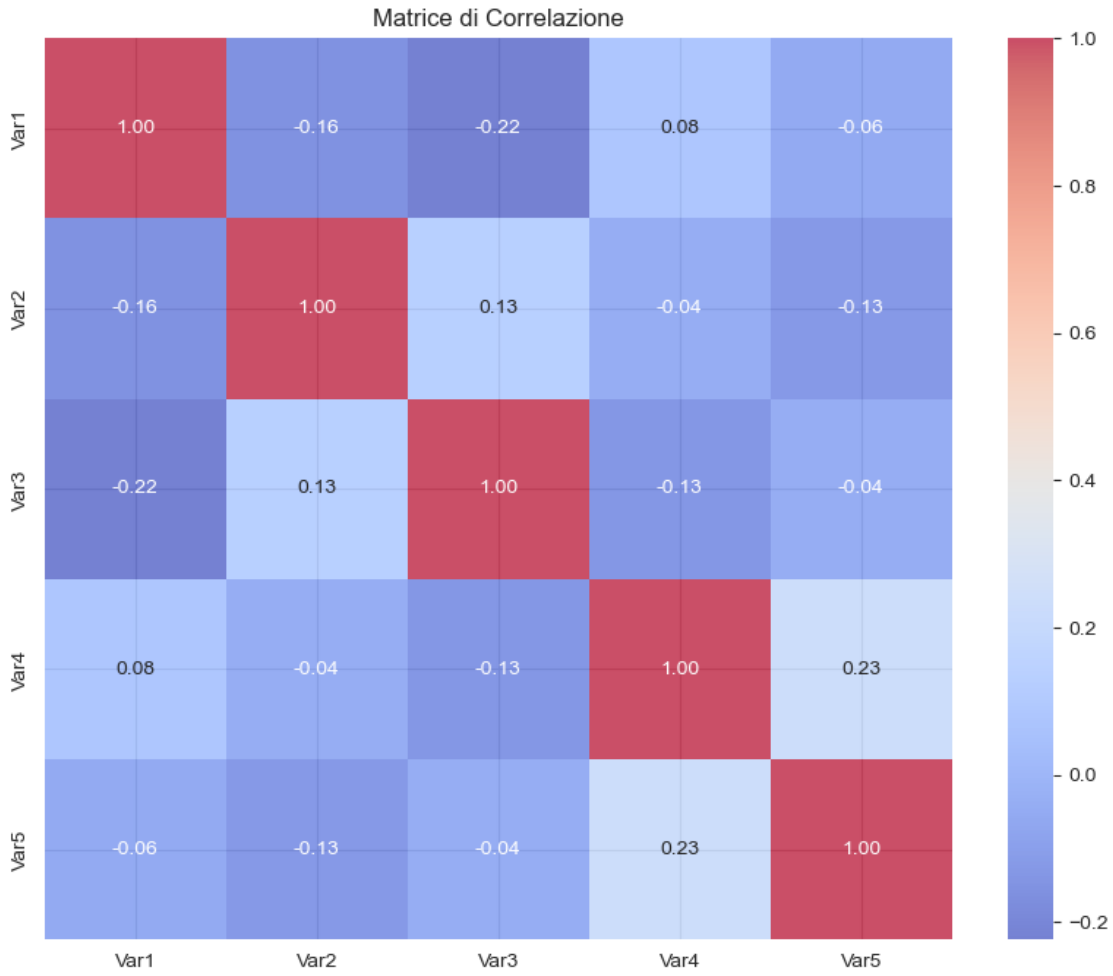
```
[7.24103329e-01, 6.37231580e-01, 9.80576800e-01, 9.03363813e-01,
 6.46664919e-01],
[6.93175211e-01, 5.15958067e-02, 6.69917465e-01, 4.41654377e-02,
 5.88263757e-01],
[9.99353500e-01, 5.55806581e-01, 4.73210118e-01, 3.12854857e-01,
 1.20522579e-01],
[7.31023671e-01, 1.92599028e-01, 1.15282021e-01, 4.21762660e-01,
 7.95341838e-01],
[7.45064548e-01, 5.48826146e-02, 4.55143685e-01, 5.22434576e-01,
 6.44546651e-01],
[6.50167350e-01, 3.64697249e-01, 5.60677194e-01, 4.81061884e-01,
 8.84998357e-01],
[5.30265133e-01, 4.41032162e-01, 4.04485450e-01, 5.72427759e-01,
 8.03976979e-01],
[5.38401490e-01, 6.61655175e-01, 7.37855767e-01, 5.20168110e-01,
 4.26817057e-01],
[8.76665378e-01, 4.17506084e-01, 4.61942851e-01, 9.90341313e-01,
 2.37523576e-04],
[1.85472910e-01, 3.83963890e-01, 9.32589909e-01, 7.00297057e-02,
 9.48543363e-03],
[5.26838360e-02, 8.85434106e-02, 3.75967732e-02, 4.79781327e-01,
 5.80757207e-01],
[2.71686971e-01, 3.98287384e-01, 9.16993351e-02, 3.36370464e-01,
 5.22519192e-01],
[7.32271807e-01, 3.34636204e-03, 4.67310767e-01, 2.97002876e-01,
 8.52821822e-01],
[7.15553315e-01, 5.89108534e-01, 2.77265646e-01, 9.10339102e-01,
 4.51006772e-02],
[1.09467599e-01, 3.91630141e-01, 1.24437557e-01, 9.56415698e-01,
 7.97989322e-01],
[2.58982652e-01, 5.88554229e-01, 9.82330515e-01, 8.84198978e-01,
 6.00798446e-01],
[9.03723967e-01, 9.89184910e-01, 7.43352823e-01, 6.49864095e-02,
 4.01580236e-01],
[8.37825735e-01, 2.30569196e-01, 8.30642518e-01, 1.20338762e-01,
 4.69453571e-02],
[3.84090611e-01, 3.67319374e-02, 9.57270263e-01, 8.26205696e-01,
 8.00769687e-01],
[6.29777923e-01, 2.16324023e-01, 5.18260367e-01, 5.97274072e-01,
 5.25031501e-01],
[2.60233368e-01, 5.15247924e-01, 4.91186341e-01, 9.96335776e-01,
 8.97436148e-01],
[4.63380748e-01, 6.22937383e-01, 7.47871604e-01, 3.48651602e-02,
 8.94921441e-01],
[8.60126816e-01, 4.57432098e-01, 3.91169766e-01, 2.72959375e-01,
 4.76676828e-01],
[4.43766635e-02, 8.52193533e-01, 3.54223684e-02, 3.29051365e-01,
```

```
 9.31589488e-01],
[7.35212655e-01, 2.81390961e-01, 2.52649094e-01, 5.08213982e-03,
 6.95714293e-01],
[4.87835817e-02, 5.32602519e-01, 9.62121973e-01, 1.09468860e-01,
 8.94479931e-01],
[9.91475335e-01, 6.15170262e-02, 8.82906269e-01, 5.16079020e-01,
 9.06513719e-01],
[5.71318942e-01, 6.62112010e-01, 5.43778908e-01, 8.64492880e-01,
 7.33909919e-01],
[5.21260312e-01, 8.68160804e-01, 2.45285532e-01, 1.57340660e-01,
 1.60400105e-01],
[3.34008879e-01, 2.38690031e-01, 9.30090351e-01, 1.05834094e-01,
 6.98990738e-01],
[1.89221540e-01, 8.81637133e-01, 9.61315288e-01, 4.65470009e-01,
 8.79071480e-01],
[5.14517075e-03, 8.19946917e-01, 2.28625162e-01, 9.08701309e-01,
 1.12011148e-03],
[5.56181735e-01, 7.56635804e-01, 2.58688331e-01, 1.93132865e-01,
 8.40982245e-02],
[3.26433577e-01, 5.59844312e-01, 1.15843857e-01, 3.76695362e-01,
 9.07240465e-03],
[8.18679420e-01, 1.28448718e-01, 2.83342948e-01, 1.84277187e-02,
 4.32240784e-01],
[8.40121503e-01, 3.92151211e-01, 1.79497323e-01, 7.53698834e-01,
 7.88806982e-01],
[1.73753475e-01, 7.70854109e-01, 9.47297888e-01, 1.66377587e-01,
 7.70984293e-01],
[7.08572005e-01, 4.73573888e-01, 5.79204606e-02, 7.46944178e-02,
 8.52414659e-01],
[9.90148373e-01, 6.76052257e-01, 4.45958307e-01, 6.47573643e-01,
 5.94395859e-01],
[3.83612260e-01, 6.56261446e-02, 8.21512947e-01, 3.71833529e-01,
 8.18785643e-01],
[2.52508750e-01, 1.02947985e-03, 1.58938283e-01, 4.68539417e-01,
 9.44303635e-01],
[1.02193620e-01, 8.54209774e-01, 5.27024096e-01, 8.25920766e-01,
 2.98374431e-01],
[8.39947165e-01, 7.72822780e-01, 2.30053683e-01, 1.31335126e-01,
 8.68054323e-01],
[9.63236653e-01, 3.13023137e-02, 7.31270067e-01, 9.09524511e-01,
 8.57360092e-01],
[9.64520472e-01, 7.60094078e-01, 4.42298762e-01, 3.11951990e-01,
 3.20742740e-01],
[7.24422673e-01, 7.45837012e-01, 2.00359144e-01, 8.68122813e-01,
 7.14924040e-01],
[5.62750286e-01, 4.71532460e-01, 5.43232806e-01, 1.59245738e-01,
 2.03701249e-01],
```

```
        [9.11644904e-02, 1.53658038e-01, 4.53693635e-01, 5.30354394e-01,
         1.78618925e-02]])
```

[45]:
```
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',␣
 ↪'Var4', 'Var5'])

# 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.
 ↪7)
plt.title("Matrice di Correlazione")
plt.show()
```

```
/var/folders/3b/sxxjhxkj11v71719m2y57znm0000gn/T/ipykernel_2041/1810228601.py:14
: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
```



4.1.37 Generazione di un DataFrame con Dati Casuali e Introduzione Casuale di Valori Mancanti

[46]:
```
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
```

[46]:

	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 x 5 columns]

4.1.38 Identificazione delle righe con dati mancanti

[47]:
```
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
len(righe_con_dati_mancanti)
```

[47]: 3648

4.1.39 Percentuale di Valori Mancanti per ciascuna Colonna nel DataFrame

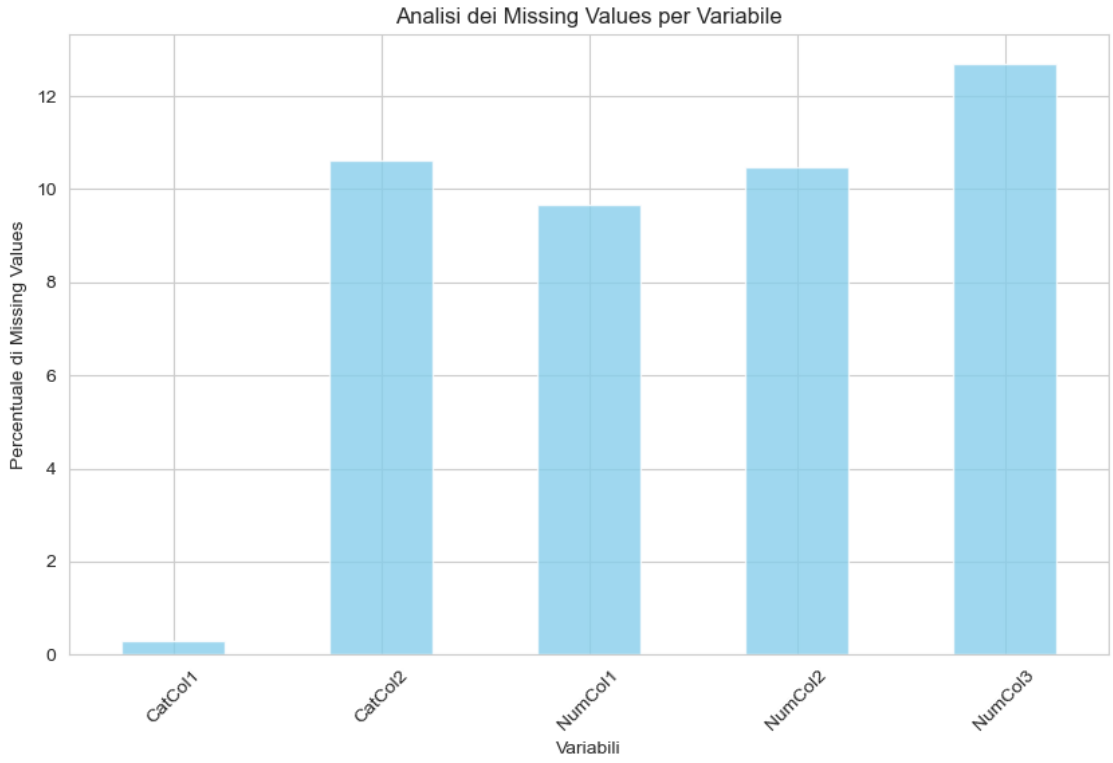
[48]:
```
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent
```

[48]:
```
CatCol1     0.29
CatCol2    10.63
NumCol1     9.67
NumCol2    10.48
NumCol3    12.69
```

```
dtype: float64
```

4.1.40 Analisi dei Missing Values per Variabile - Grafico a Barre della Percentuale di Valori Mancanti

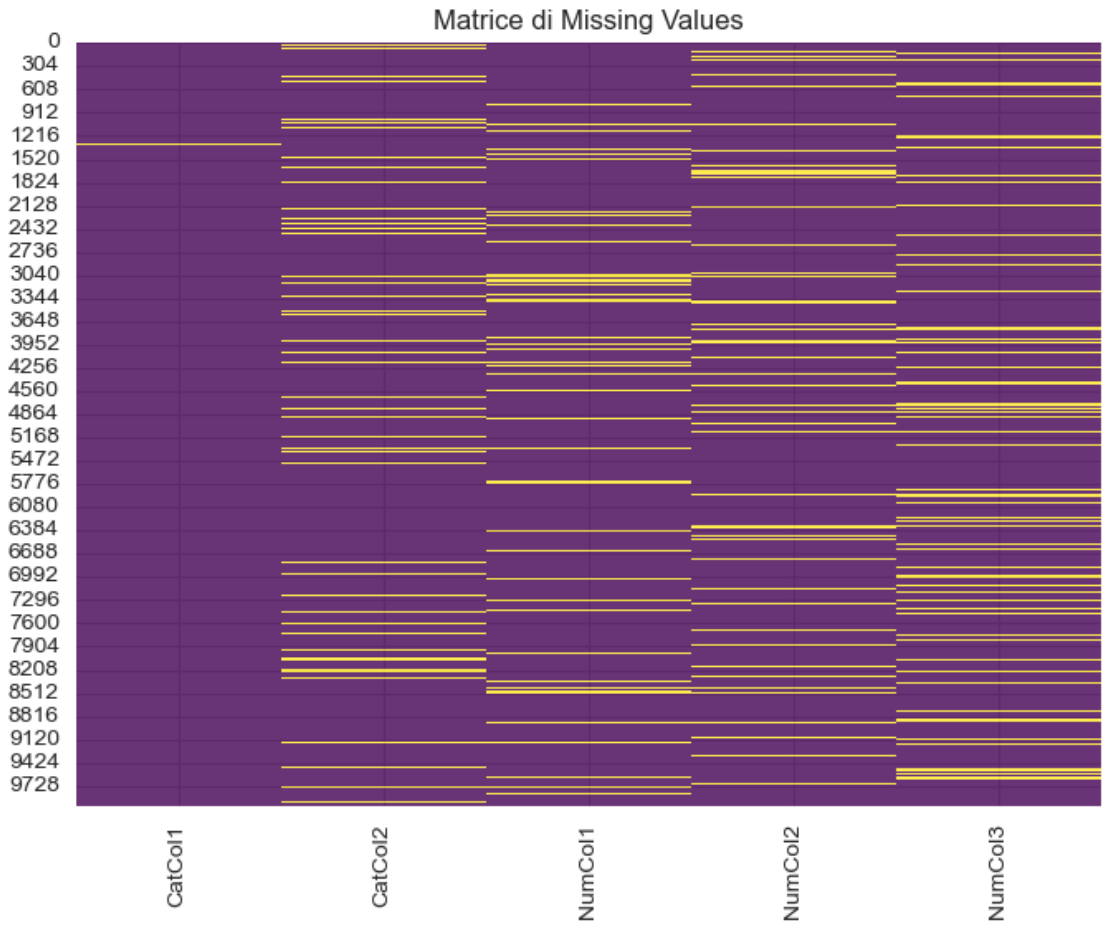
[49]:
```
# Crea il grafico a barre per la percentuale di valori mancanti
plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue', alpha=0.8)
plt.xlabel('Variabili')
plt.ylabel('Percentuale di Missing Values')
plt.title('Analisi dei Missing Values per Variabile')
plt.xticks(rotation=45)
plt.show()
```



4.1.41 Matrice di Missing Values - Heatmap della Distribuzione dei Valori Mancanti nel DataFrame

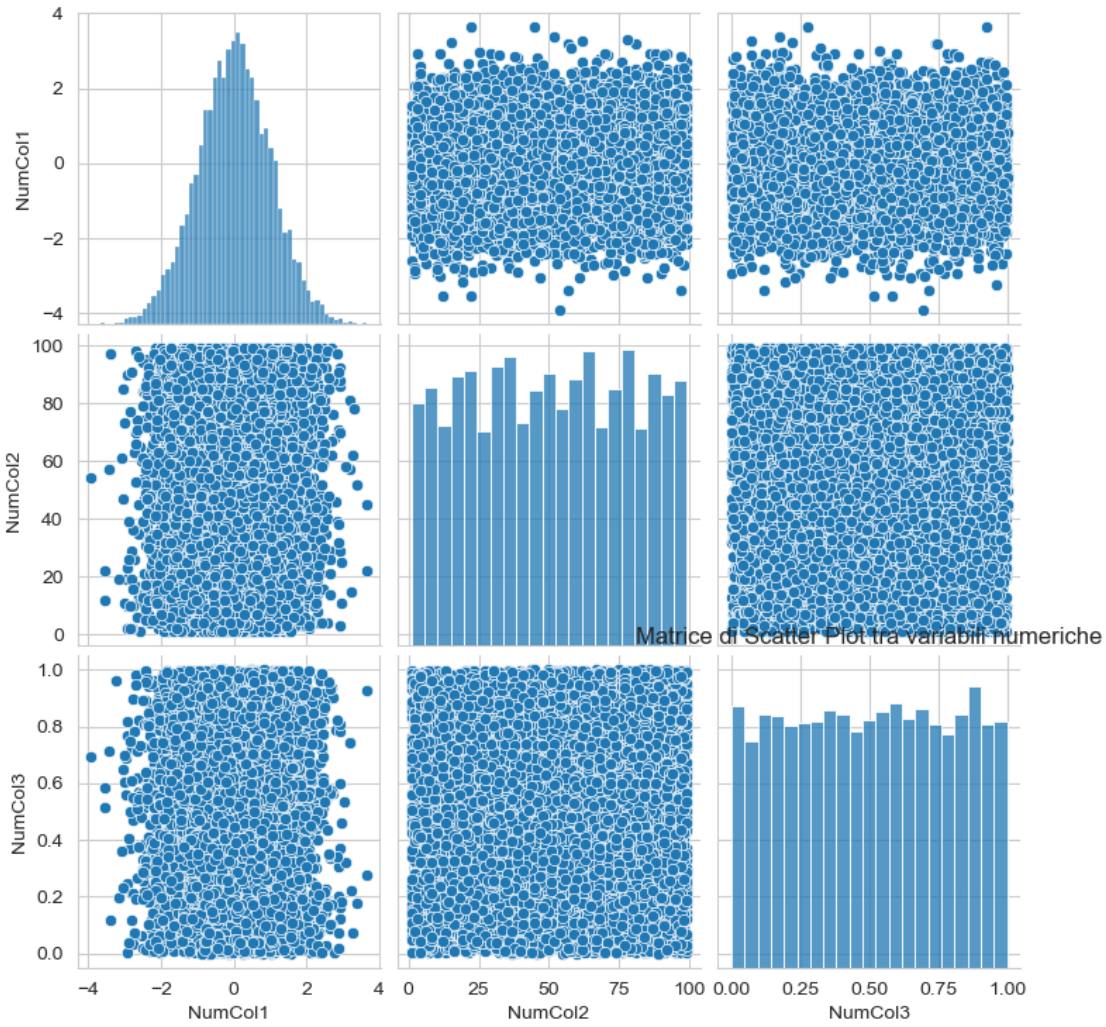
[51]:
```
# Crea la matrice booleana indicando i valori mancanti
missing_matrix = df.isnull()

# Crea una heatmap colorata per visualizzare i valori mancanti
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()
```



4.1.42 Matrice di Scatter Plot tra Variabili Numeriche nel DataFrame

[52]:
```
# Seleziona solo le 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()
```



4.1.43 Eliminazione delle Righe con Valori NaN nelle Features Categoriche del DataFrame

[53]:
```
# Elimina le righe in cui entrambe le features categoriche hanno valori NaN
df = df.dropna(subset=["CatCol1", 'CatCol2'], how='all')
df
```

[53]:

	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 x 5 columns]
```

4.1.44 Eliminazione delle Righe con Valori NaN nelle Features Numeriche del DataFrame

[55]:
```
# Elimina le righe in cui tutte le colonne numeriche specificate hanno valori NaN
df = df.dropna(subset=["NumCol1", "NumCol2", "NumCol3"], how='all')
df
```

[55]:
```
     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 x 5 columns]
```

4.1.45 Gestione dei Missing Values: Imputazione della Moda per le Colonne Categoriche e Media Condizionata per le Colonne Numeriche nel DataFrame

[56]:
```
# Seleziona le colonne numeriche e categoriche
numeric_cols = df.select_dtypes(include=['number'])
categorical_cols = df.select_dtypes(exclude=['number'])

# Sostituisci i missing values nelle colonne categoriche con la moda
df.loc[:, categorical_cols.columns] = df[categorical_cols.columns].
 ↪fillna(df[categorical_cols.columns].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].transform('mean'))

# Aggiorna le colonne numeriche con la media condizionata
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]
```

```
/var/folders/3b/sxxjhxkj11v71719m2y57znm0000gn/T/ipykernel_2041/1186182115.py:6:
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

/var/folders/3b/sxxjhxkj11v71719m2y57znm0000gn/T/ipykernel_2041/1186182115.py:12
: 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
```

4.1.46 Gestione dei Missing Values in un DataFrame Casuale con Variabili Categoriche e Numeriche

[61]:
```
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␣
 ↪.loc
df.loc[:, categorical_cols.columns] = df[categorical_cols.columns].
 ↪fillna(df[categorical_cols.columns].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].transform('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]
```