rappresenta i dataset con i dati mancanti.

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

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

Out[1]:

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

df = pd.DataFrame(dataset): Crea un DataFrame pandas utilizzando la lista di dizionari.

Continuo codice di prima

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

Out[2]:

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

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

Out[3]:

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

conta quante righe con dati mancanti ci sono in totale

In [42]:
```
totale_dati_mancanti = righe_con_dati_mancanti.shape[0]
totale_dati_mancanti
#righe_con_dati_mancanti.shape: Questo restituisce una tupla che rappresenta le dimensioni del DataFrame righe_
#righe_con_dati_mancanti.shape[0]: Prendiamo il primo elemento della tupla restituita da shape, che rappresenta
```

Out[42]:
```
3648
```

crea email con i dataset.

In [4]:
```
import pandas as pd

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

```
    {"nome": "Charlie", "età": 28, "punteggio": 75, "email": "charlie@email.com"},
]

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

Out[4]:

	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

rimuove le righe con i dati mancanti e mdoifica il dataframe (codice sopra)

In [5]:
```
# Rimuovi le righe con dati mancanti e modifica il DataFrame originale
df.dropna(inplace=False)
df
```

Out[5]:

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

In [6]:
```
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() #df1 = pd.DataFrame(): Crea un DataFrame vuoto chiamato df1. Al momento dell'esecuzione, que
df
```

Out[6]:

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

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

Out[11]:

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

In [12]:
```
# Trattamento dei missing values nelle variabili categoriche
categorical_cols = df.select_dtypes(exclude=['number'])
categorical_cols.columns
```

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

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

Out[13]:

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

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

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

Out[14]:

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

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
Feature1    1
Feature2    2
Feature3    1
dtype: int64
```

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

Out[16]:

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

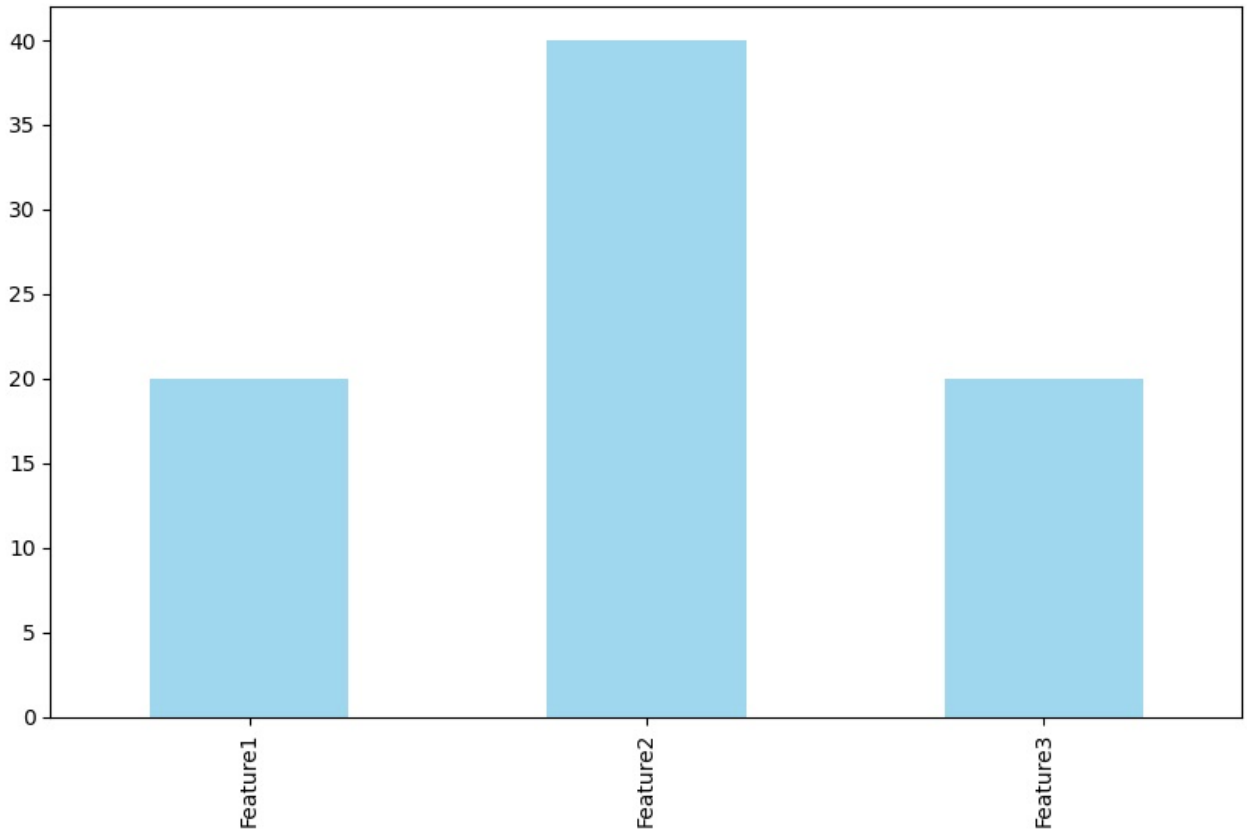
moltiplicazione per 100 per ottenere la percentuale dei valori mancanti, divisione per la proposizione dei valori mancanti

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

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

creazione grafico a barre con dentro le missing values

In [18]:
```
plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue', alpha=0.8)
plt.show()
```



In [19]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

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

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

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

Out[19]:

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

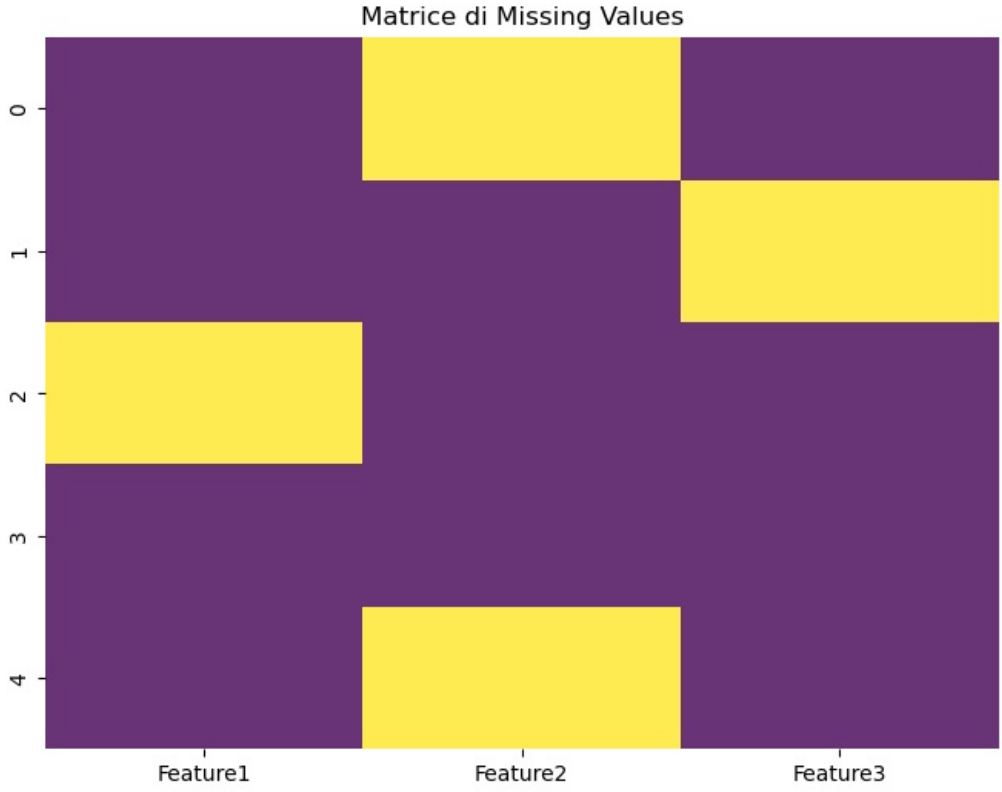
In [43]:
```
missing_matrix = df.isnull()
missing_matrix
```

Out[43]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	False	True	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
9995	False	False	False	False	False
9996	False	False	False	False	False
9997	False	False	False	False	False
9998	False	False	False	True	True
9999	False	False	False	False	False

10000 rows × 5 columns

In [20]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(missing_matrix, cmap='viridis', cbar=False,alpha=0.8)
plt.title('Matrice di Missing Values')
plt.show()
```



In [21]:
```
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'esplorazione
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())
```

```
   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
```

da Informazioni sul dataset

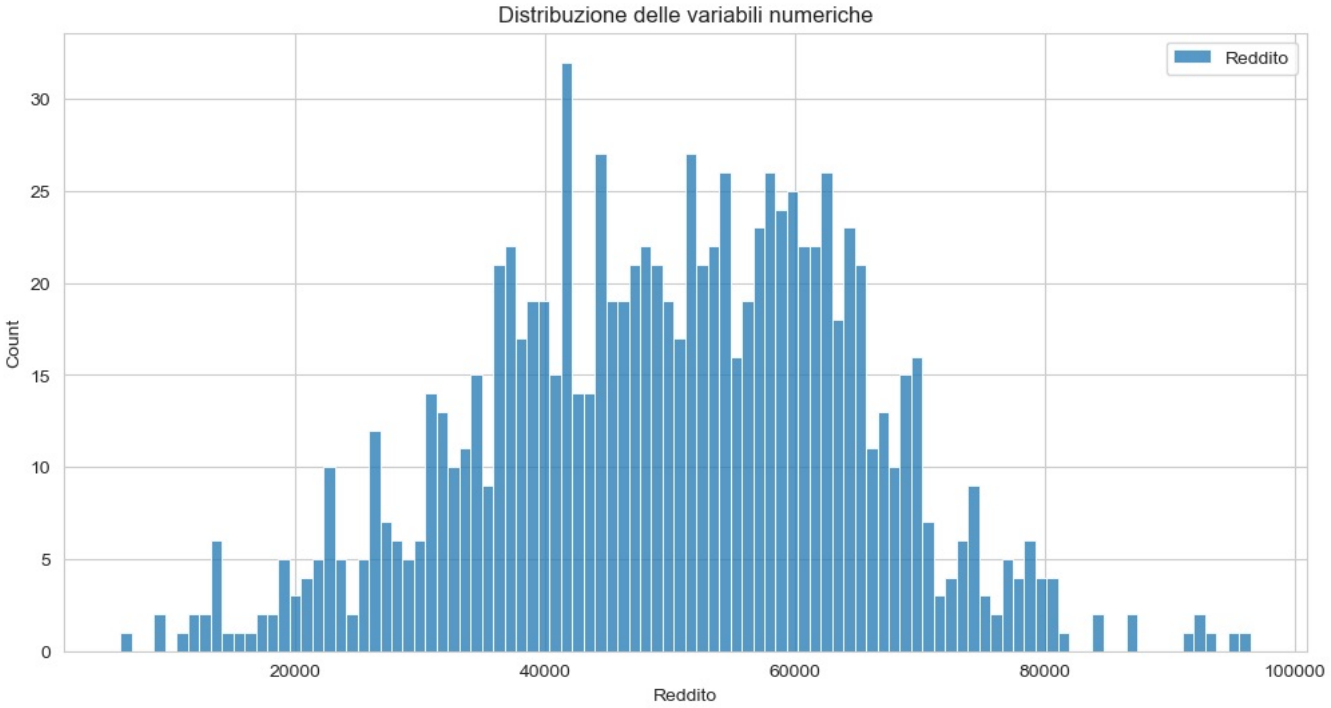
In [22]:
```
print(df.info())

#Statistiche descrittive
print(df.describe())
#df.info(): Questo metodo restituisce una sintesi delle informazioni sul DataFrame df. Include il numero totale
```

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

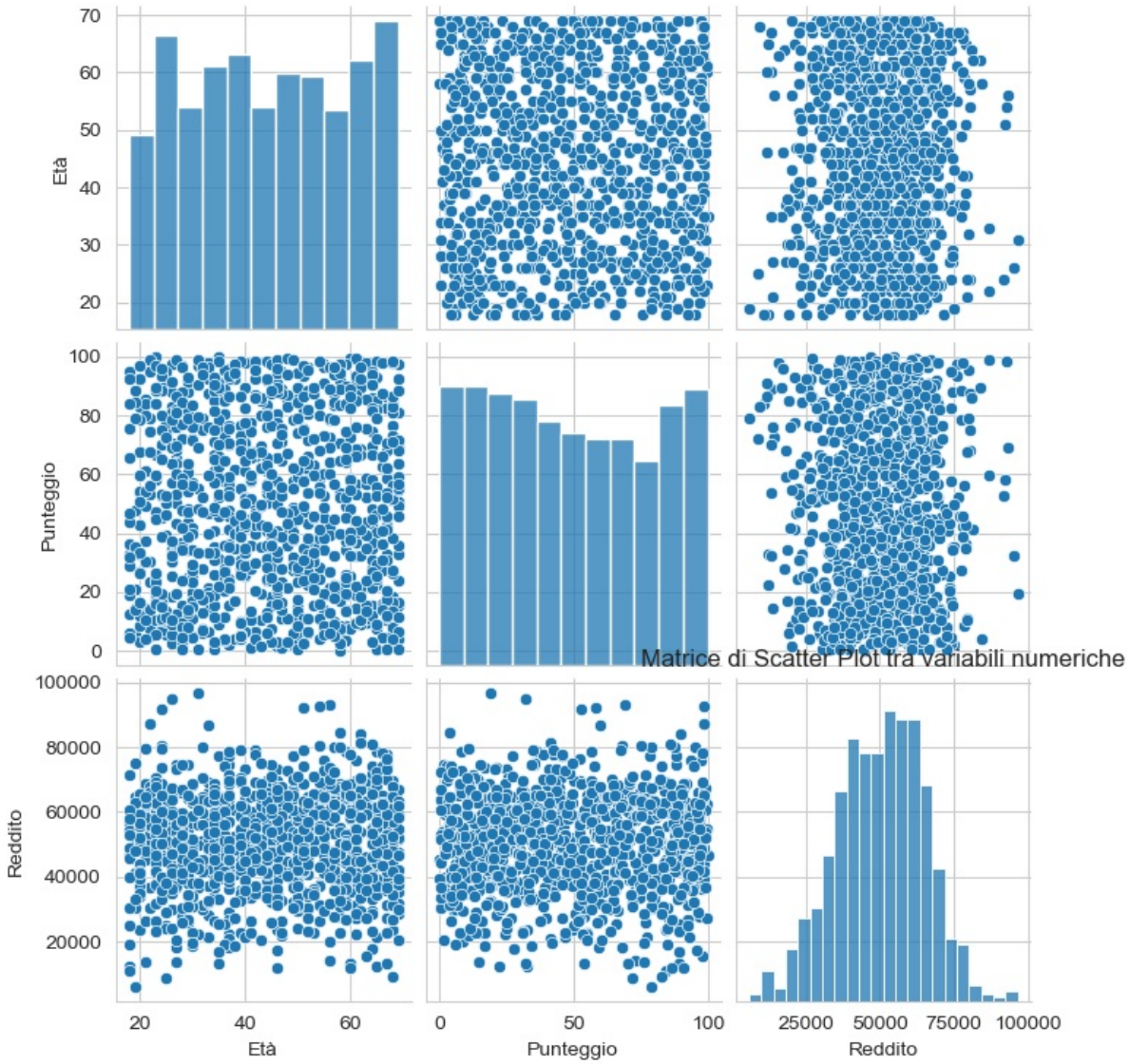
In [23]:
```
missing_data = df.isnull().sum()
print('Valori mancanti per ciascuna colonna:')
print(missing_data)
```

```
Valori mancanti per ciascuna colonna:
Età          0
Genere       0
Punteggio    0
Reddito      0
dtype: int64
```

In [25]:
```
plt.figure(figsize=(12, 6))
sns.set_style("whitegrid")
sns.histplot(df["Reddito"], kde=False, bins=100, label="Reddito")
plt.legend()
plt.title('Distribuzione delle variabili numeriche')
plt.show()
```

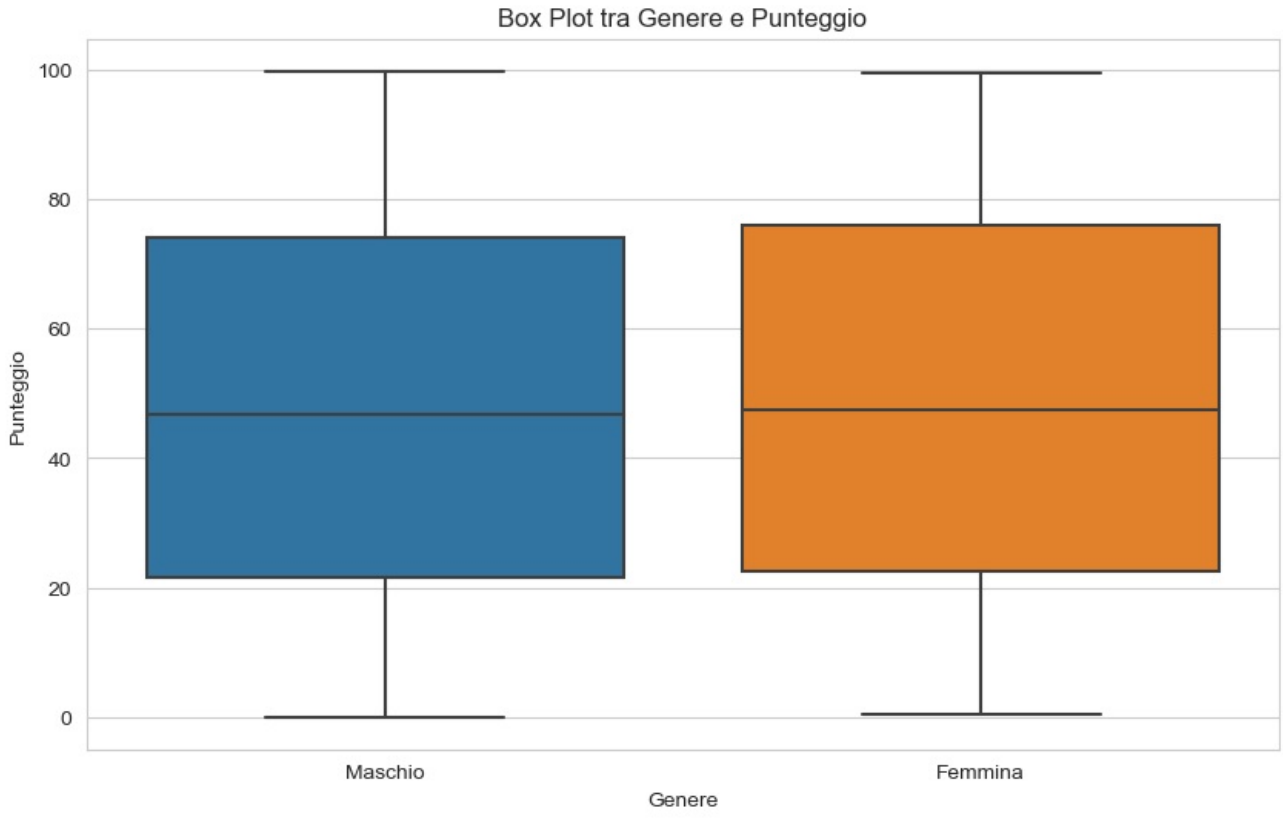


In [26]:
```
numeric_features = df.select_dtypes(include=[np.number])
sns.pairplot(df[numeric_features.columns])
plt.title("Matrice di Scatter Plot tra variabili numeriche")
plt.show()
```



Matrice di Scatter Plot tra variabili numeriche

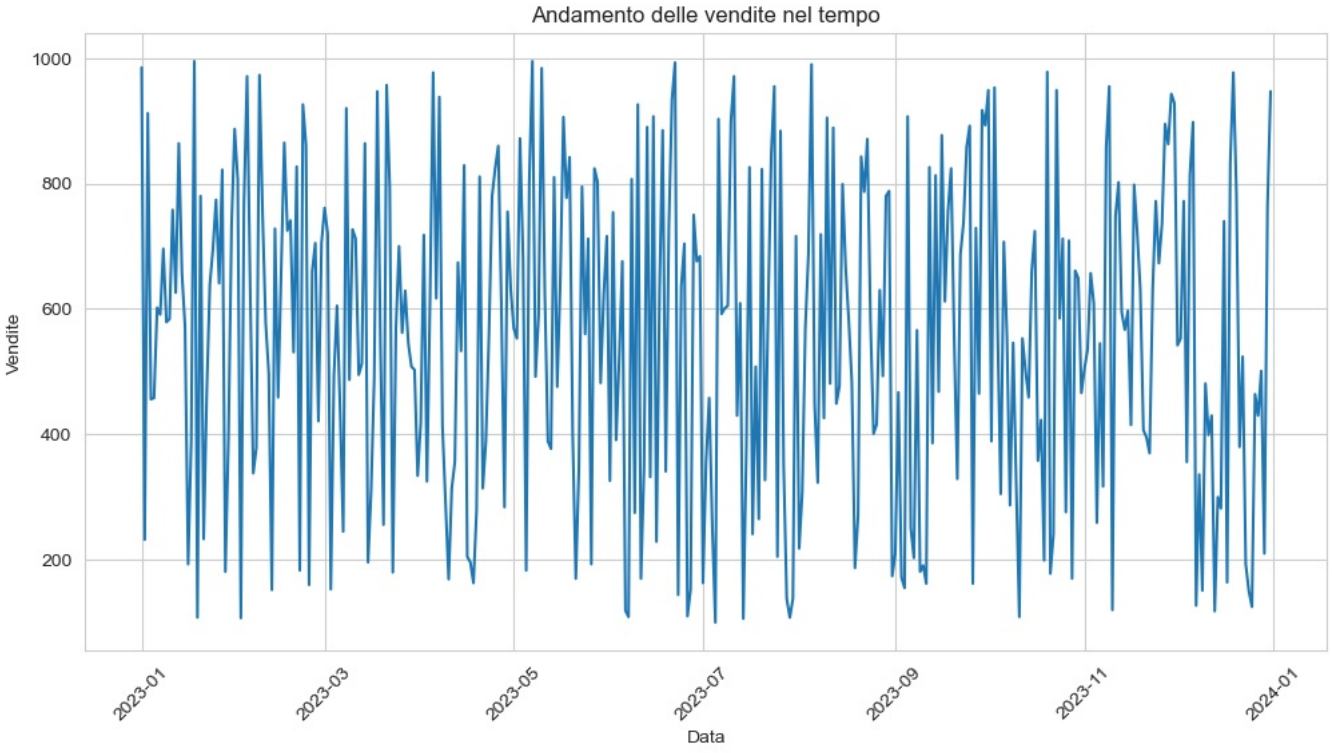
In [27]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x="Genere", y="Punteggio", data=df)
plt.title("Box Plot tra Genere e Punteggio")
plt.show()
```

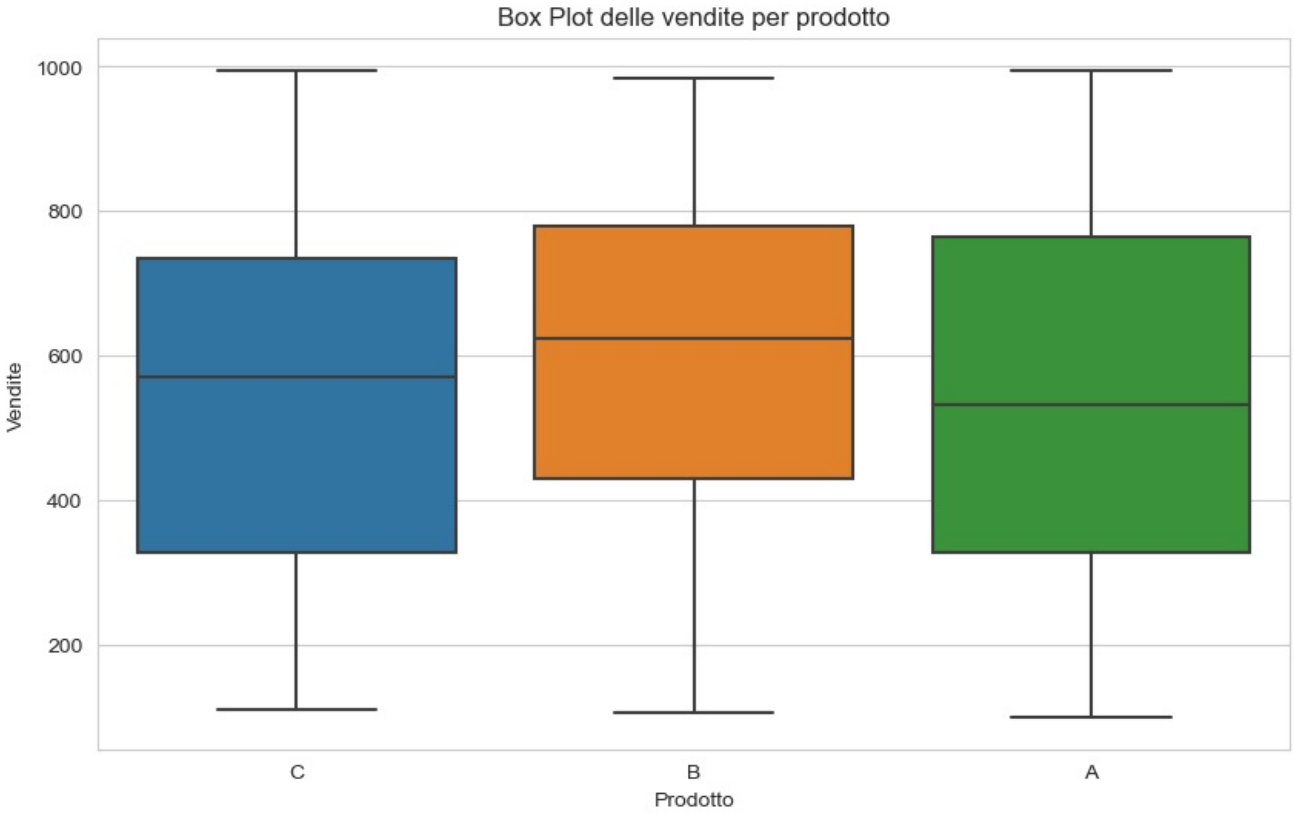


In [28]:
```
import pandas as pd
```

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

df = pd.DataFrame(data)

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

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

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

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





In [29]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

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

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

#import pandas as pd: Importa la libreria pandas con l'alias 'pd', che è comunemente utilizzata per la manipola
#import matplotlib.pyplot as plt: Importa il modulo pyplot da Matplotlib con l'alias 'plt', che è spesso utiliz
#import numpy as np: Importa la libreria numpy con l'alias 'np', che è comunemente utilizzata per operazioni ma
#import seaborn as sns: Importa la libreria seaborn con l'alias 'sns', che è spesso utilizzata per la visualizz
#data = {...}: Definisce un dizionario chiamato data contenente due chiavi ('Numeric_Var' e 'Categorical_Var')
#df = pd.DataFrame(data): Crea un DataFrame pandas utilizzando il dizionario data. Ogni chiave diventa una colo
#print(df): Stampa il DataFrame per visualizzare i dati di esempio. Questo è utile per esaminare r
```

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

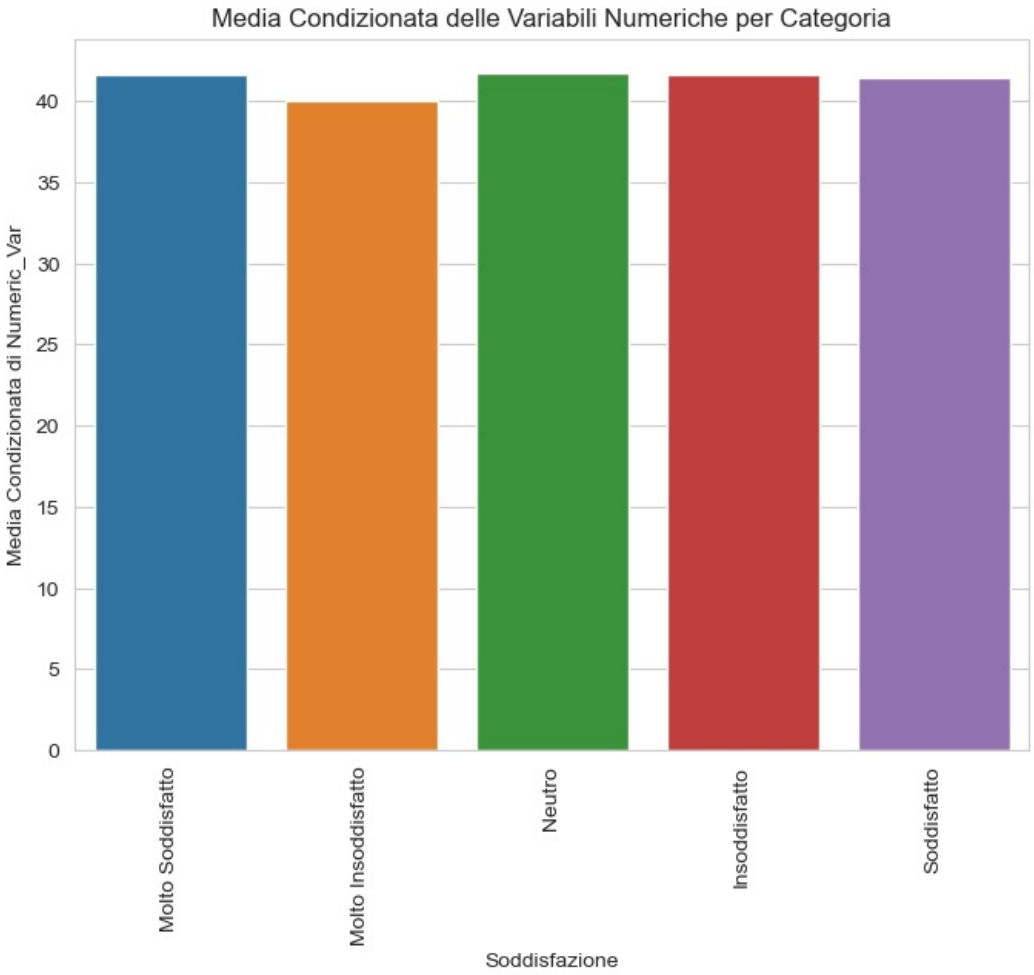
In [30]:
```python
conditional_means = df['Numeric_Var'].fillna(df.groupby('Categorical_Var')['Numeric_Var'].transform('mean'))
```

In [31]:
```python
import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

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

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

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

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

plt.show()
```

```
     Età        Soddisfazione
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]
```

```
C:\Users\alberto\AppData\Local\Temp\ipykernel_2352\2447909657.py:23: FutureWarning:

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

  sns.barplot(data=df, x='Soddisfazione', y='Numeric_Var', ci=None)
```



In [32]:
```python
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
#np.random.seed(41): Imposta il seed del generatore di numeri casuali a 41 per garantire la riproducibilità dei
#df = pd.DataFrame(): Crea un DataFrame pandas vuoto chiamato df.
#n_rows = 10000: Imposta il numero di righe del DataFrame a 10000.
#df['CatCol1'] = np.random.choice(['A', 'B', 'C'], size=n_rows): Aggiunge una colonna 'CatCol1' al DataFrame co
#df['CatCol2'] = np.random.choice(['X', 'Y'], size=n_rows): Aggiunge una colonna 'CatCol2' al DataFrame contene
#df['NumCol1'] = np.random.randn(n_rows): Aggiunge una colonna 'NumCol1' al DataFrame contenente valori casuali
#df['NumCol2'] = np.random.randint(1, 100, size=n_rows): Aggiunge una colonna 'NumCol2' al DataFrame contenente
#df['NumCol3'] = np.random.uniform(0, 1, size=n_rows): Aggiunge una colonna 'NumCol3' al DataFrame
```

Out[32]:

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

10000 rows × 5 columns

In [33]:
```
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
righe_con_dati_mancanti
```

Out[33]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
5	B	NaN	NaN	71.0	0.752397
6	B	X	0.080686	31.0	NaN
8	B	Y	-1.291483	NaN	0.868791
12	C	Y	-1.193705	8.0	NaN
...	...	...	...	...	...
9986	C	X	-0.909994	NaN	0.767918
9988	A	Y	NaN	35.0	0.149513
9989	A	NaN	-0.148047	NaN	0.326089
9992	A	Y	-0.048300	58.0	NaN
9998	C	Y	0.004278	NaN	NaN

3648 rows × 5 columns

In [34]:
```
missing_percent = (df.isnull() / len(df)) * 100
missing_percent
```

Out[34]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	0.0	0.01	0.0	0.00	0.00
1	0.0	0.00	0.0	0.00	0.00
2	0.0	0.00	0.0	0.00	0.00
3	0.0	0.00	0.0	0.00	0.00
4	0.0	0.00	0.0	0.00	0.00
...	...	...	...	...	...
9995	0.0	0.00	0.0	0.00	0.00
9996	0.0	0.00	0.0	0.00	0.00
9997	0.0	0.00	0.0	0.00	0.00
9998	0.0	0.00	0.0	0.01	0.01
9999	0.0	0.00	0.0	0.00	0.00

10000 rows × 5 columns

In [35]:
```
df.isnull().sum()/100
```

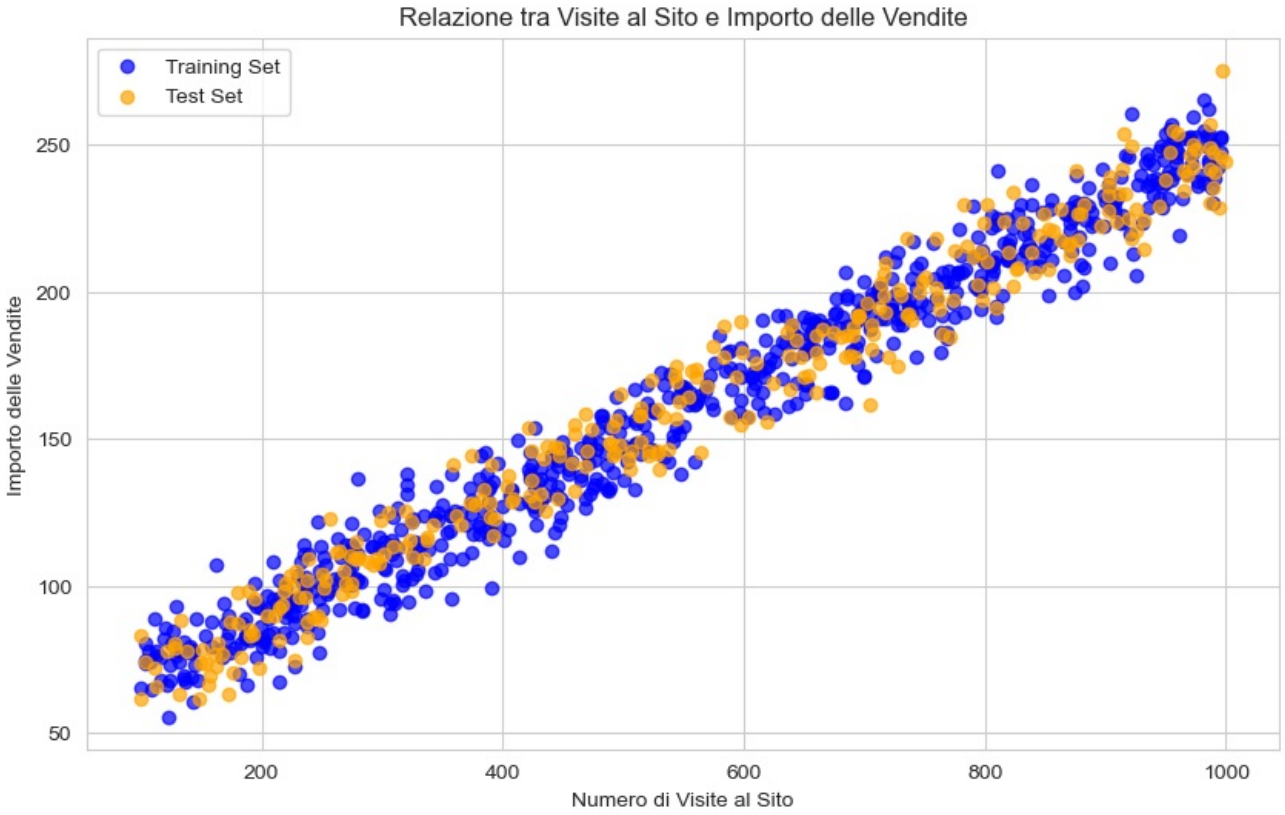
Out[35]:
```
CatCol1     0.29
CatCol2    10.63
NumCol1     9.67
NumCol2    10.48
NumCol3    12.69
dtype: float64
```

In [36]:
```
import numpy as np
from sklearn.model_selection import train_test_split

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

# Suddividere il dataset in training set (70%) e test set (30%)
X_train, X_test, y_train, y_test = train_test_split(altezze, pesi, test_size=0.3, random_state=42)

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

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

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

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

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

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

# Stampare le dimensioni dei training set e test set
print("Dimensioni del Training Set (visite al sito e importo delle vendite):", X_train.shape, y_train.shape)
print("Dimensioni del Test Set (visite al sito e importo delle vendite):", X_test.shape, y_test.shape)
#import numpy as np: Importa la libreria NumPy e la assegna all'alias np
#import matplotlib.pyplot as plt: Importa il modulo pyplot da Matplotlib per la creazione di grafici.
#from sklearn.model_selection import train_test_split: Importa la funzione train_test_split da scikit-learn per
#np.random.seed(0): Imposta il seed del generatore di numeri casuali di NumPy a 0 per garantire la riproducibil
#visite_al_sito = np.random.randint(100, 1000, 1000): Genera un array di 1000 numeri interi casuali compresi tr
#importo_vendite = 50 + 0.2 * visite_al_sito + np.random.normal(0, 10, 1000): Genera un array di importi di ven
#X_train, X_test, y_train, y_test = train_test_split(visite_al_sito, importo_vendite, test_size=0.3, random_sta
#plt.figure(figsize=(10, 6)): Crea una nuova figura per il grafico con una dimensione di larghezza 10 pollici e
#plt.scatter(X_train, y_train, label='Training Set', color='blue', alpha=0.7): Crea uno scatter plot per i dati
#plt.scatter(X_test, y_test, label='Test Set', color='orange', alpha=0.7): Crea uno scatter plot per i dati di
#plt.xlabel('Numero di Visite al Sito'): Aggiunge un'etichetta all'asse x indicando il numero di visite al sito
#plt.ylabel('Importo delle Vendite'): Aggiunge un'etichetta all'asse y indicando l'importo delle vendite.
#plt.title('Relazione tra Visite al Sito e Importo delle Vendite'): Aggiunge un titolo al grafico descrivendo l
```

```
#plt.legend(): Aggiunge una legenda al grafico per distinguere i dati di training e test.
#plt.grid(True): Aggiunge una griglia al grafico per migliorare la leggibilità.
#plt.show(): Mostra il grafico.
#print("Dimensioni del Training Set (visite al sito e importo delle vendite):", X_train.shape, y_train.shape): 
#print("Dimensioni del Test Set (visite al sito e importo delle vendite):", X_test.shape, y_test.shape): Stampa
```



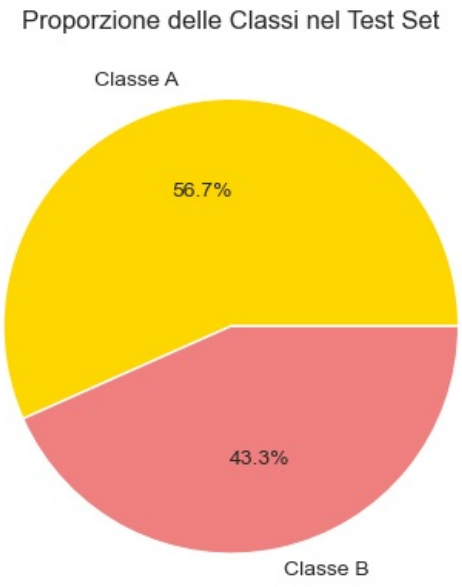

```
Dimensioni del Training Set (visite al sito e importo delle vendite): (700,) (700,)
Dimensioni del Test Set (visite al sito e importo delle vendite): (300,) (300,)
```

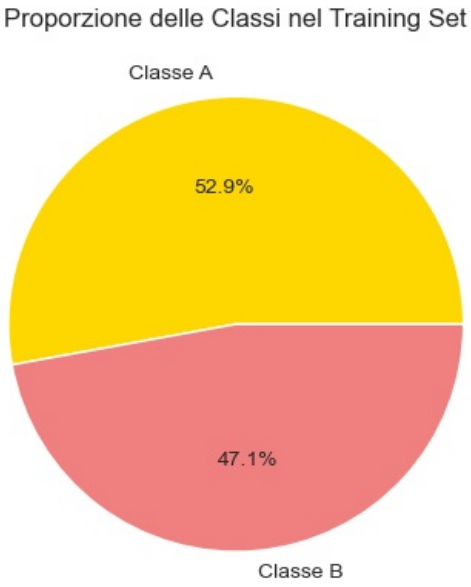
In [38]:
```
from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(1)
# Supponiamo di avere un dataset con feature X e target y
X = np.random.rand(100, 2)  # Dati del dataset (100 campioni, 2 feature)
y = np.random.choice(['A', 'B'], size=100)  # Etichette di classe casuali
# Calcola le proporzioni delle classi nel dataset originale
proporzione_classe_A = sum(y == 'A') / len(y)
proporzione_classe_B = 1 - proporzione_classe_A
# Eseguire uno split stratificato con una proporzione specificata
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Calcola le proporzioni delle classi nel training set e nel test set
proporzione_classe_A_train = sum(y_train == 'A') / len(y_train)
proporzione_classe_B_train = 1 - proporzione_classe_A_train

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

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

```
Proporzione Classe A nel data Set completo: 0.54
Proporzione Classe B nel data Setcompleto: 0.45999999999999996
Proporzione Classe A nel Training Set: 0.5285714285714286
Proporzione Classe B nel Training Set: 0.4714285714285714
Proporzione Classe A nel Test Set: 0.5666666666666667
Proporzione Classe B nel Test Set: 0.43333333333333335
```

In [39]:
```
labels=['Classe A', 'Classe B']
colors=['gold', 'lightcoral']
plt.pie([proporzione_classe_A_test, proporzione_classe_B_test], labels=labels, colors=colors, autopct='%1.1f%%'
plt.title('Proporzione delle Classi nel Test Set')
plt.show()

labels=['Classe A', 'Classe B']
colors=['gold', 'lightcoral']
plt.pie([proporzione_classe_A_train, proporzione_classe_B_train], labels=labels, colors=colors, autopct='%1.1f%
plt.title('Proporzione delle Classi nel Training Set')
plt.show()
#labels=['Classe A', 'Classe B']: Definisce le etichette per le due classi.
```

```
#colors=['gold', 'lightcoral']: Definisce i colori da utilizzare nei grafici a torta per le due classi.
#plt.pie([proporzione_classe_A_test, proporzione_classe_B_test], labels=labels, colors=colors, autopct='%1.1f%%
#plt.title('Proporzione delle Classi nel Test Set'): Aggiunge un tito
```





In [ ]: