OUTLIERS

1-Divisione dei Dati Altezza-Peso per Addestramento e Test

In [5]:

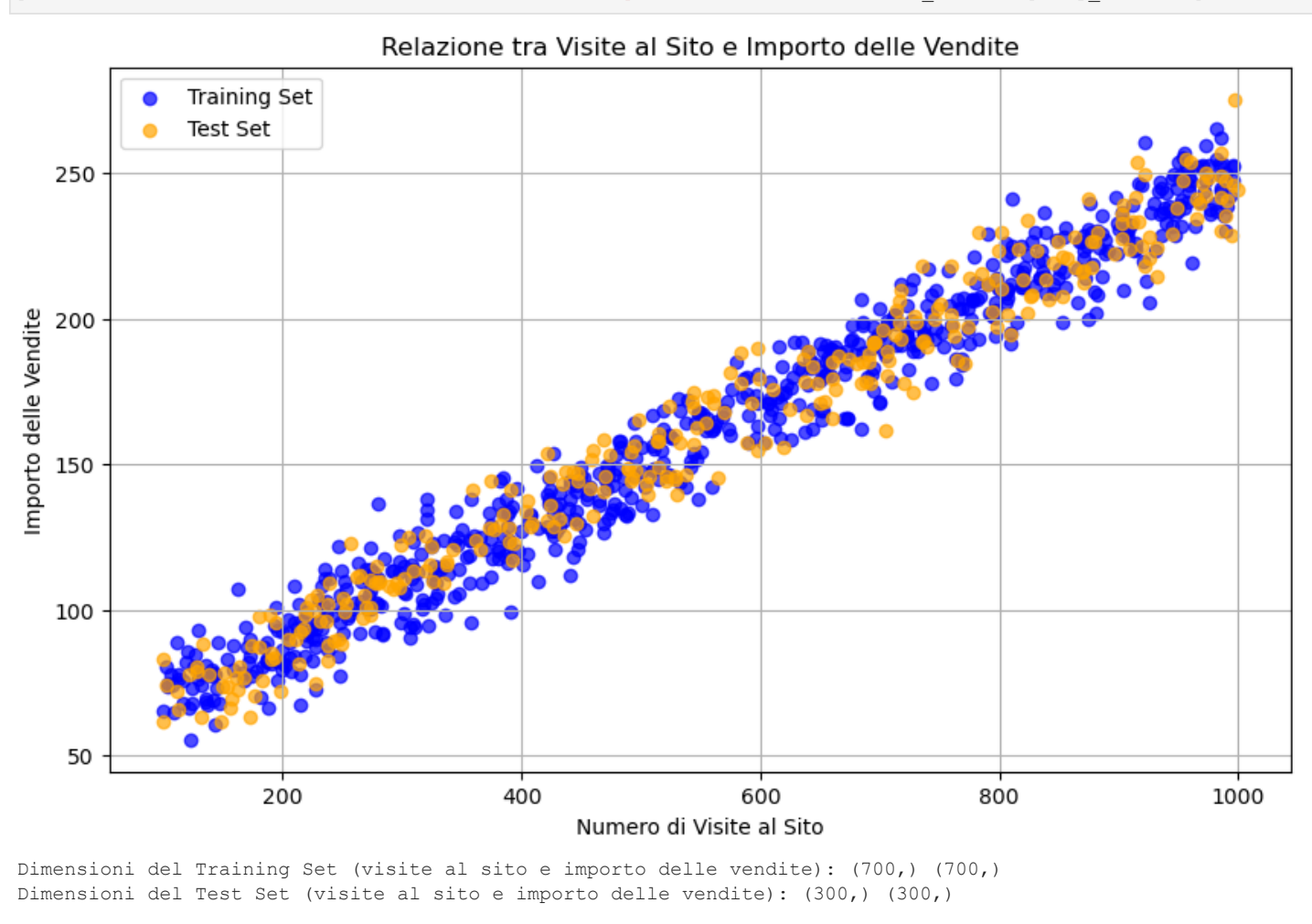
```
# Importa la libreria NumPy con l'alias "np"
import numpy as np
# Importa la funzione train_test_split dal modulo model_selection della libreria scikit-learn
from sklearn.model_selection import train_test_split
# Imposta il seme casuale a 0 per la riproducibilità dei risultati
np.random.seed(0)
# Genera 100 campioni casuali da una distribuzione normale con media 0 e deviazione standard 5, rappresentanti le altezze
altezze = np.random.normal(0, 5, 100)
# Genera i pesi corrispondenti alle altezze, dove il peso è calcolato come metà dell'altezza più un termine di errore casuale
pesi = 0.5 * altezze + np.random.normal(0, 5, 100)
# Suddivide le altezze e i pesi in set di addestramento e di test utilizzando la funzione train_test_split
# Il 30% dei dati è assegnato al set di test, il restante 70% al set di addestramento
# random_state=42 garantisce la riproducibilità dei risultati
X_train, X_test, y_train, y_test = train_test_split(altezze, pesi, test_size=0.3, random_state=42)
# Stampa le dimensioni del set di addestramento, che consiste nelle altezze e nei pesi
print("Dimensioni del Training Set (altezze e pesi):", X_train.shape, y_train.shape)
# Stampa le dimensioni del set di test, che consiste nelle altezze e nei pesi
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,)
```

2-Analisi della Relazione tra Visite al Sito e Importo delle Vendite

In [6]:

```
# Importa le librerie necessarie
import numpy as np  # Importa NumPy per la generazione di dati casuali
import matplotlib.pyplot as plt  # Importa Matplotlib per la visualizzazione grafica
from sklearn.model_selection import train_test_split  # Importa train_test_split per la suddivisione del dataset
# Imposta il seme casuale per la riproducibilità dei risultati
np.random.seed(0)
# Genera dati casuali per il numero di visite al sito web e l'importo delle vendite
visite_al_sito = np.random.randint(100, 1000, 1000)  # Genera 1000 numeri casuali interi tra 100 e 1000
importo_vendite = 50 + 0.2 * visite_al_sito + np.random.normal(0, 10, 1000)  # Calcola l'importo delle vendite con una relazione lineare rispetto alle visite al sito
# Suddivide il dataset in set di addestramento (70%) e set di test (30%)
X_train, X_test, y_train, y_test = train_test_split(visite_al_sito, importo_vendite, test_size=0.3, random_state=42)
# Crea un grafico a dispersione per visualizzare la relazione tra visite al sito e importo delle vendite
plt.figure(figsize=(10, 6))  # Imposta le dimensioni della figura
plt.scatter(X_train, y_train, label='Training Set', color='blue', alpha=0.7)  # Grafico a dispersione per il set di addestramento
plt.scatter(X_test, y_test, label='Test Set', color='orange', alpha=0.7)  # Grafico a dispersione per il set di test
plt.xlabel('Numero di Visite al Sito')  # Etichetta dell'asse x
plt.ylabel('Importo delle Vendite')  # Etichetta dell'asse y
plt.title('Relazione tra Visite al Sito e Importo delle Vendite')  # Titolo del grafico
plt.legend()  # Mostra la legenda
plt.grid(True)  # Mostra la griglia
plt.show()  # Mostra il grafico
# Stampa le dimensioni dei set di addestramento e test
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)
```



```
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,)
```

3-Analisi delle Proporzioni delle Classi nei Set di Addestramento e Test

In [1]:

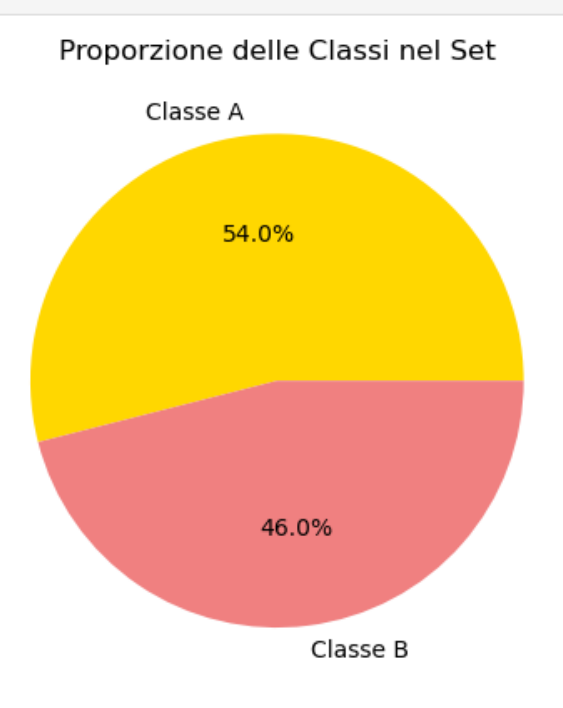
```
from sklearn.model_selection import train_test_split  # Importa la funzione train_test_split da sklearn
import numpy as np  # Importa NumPy per la generazione di dati casuali

np.random.seed(1)  # Imposta il seme casuale per la riproducibilità dei risultati

# Genera 100 campioni casuali distribuiti uniformemente tra 0 e 1 per due caratteristiche (variabili)
X = np.random.rand(100, 2)

# Genera 100 etichette casuali "A" o "B"
Y = np.random.choice(['A', 'B'], size=100)

# Calcola la proporzione della classe 'A' nel vettore delle etichette
proporzione_classe_A = sum(Y == 'A') / len(Y)

# Calcola la proporzione della classe 'B' nel vettore delle etichette
proporzione_classe_B = 1 - proporzione_classe_A

# Suddivide il dataset in set di addestramento (70%) e set di test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Calcola la proporzione della classe 'A' nel set di addestramento
proporzione_classe_A_train = sum(y_train == 'A') / len(y_train)

# Calcola la proporzione della classe 'B' nel set di addestramento
proporzione_classe_B_train = 1 - proporzione_classe_A_train

# Calcola la proporzione della classe 'A' nel set di test
proporzione_classe_A_test = sum(y_test == 'A') / len(y_test)

# Calcola la proporzione della classe 'B' nel set di test
proporzione_classe_B_test = 1 - proporzione_classe_A_test

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

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

4-Visualizzazione della Proporzione delle Classi nel Set

In [5]:

```
import matplotlib.pyplot as plt
# Definizione delle etichette per le classi
labels = ['Classe A', 'Classe B']

# Definizione dei colori per le fette della torta
colors = ['gold', 'lightcoral']

# Generazione del grafico a torta con le proporzioni delle classi
plt.pie([proporzione_classe_A, proporzione_classe_B], labels=labels, colors=colors, autopct='%1.1f%%')

# Aggiunta di un titolo al grafico
plt.title('Proporzione delle Classi nel Set')

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



5-Analisi Statistica di un Campione Casuale

In [6]:

```
import random  # Importa il modulo random per generare numeri casuali
import numpy as np  # Importa il modulo NumPy per calcoli numerici

dataset = []  # Inizializza una lista vuota per il dataset

# Creazione di un dataset di 1000 elementi (dati casuali compresi tra 1 e 100)
for i in range(1000):
    dataset.append(random.randint(1, 100))  # Aggiunge un numero casuale compreso tra 1 e 100 al dataset

# Estrazione di un campione casuale semplice di 300 elementi dal dataset
campione_casuale = random.sample(dataset, 300)  # Estrae 300 elementi casuali dal dataset

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

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

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

```
Media del campione casuale: 50.57
Deviazione standard del campione casuale: 28.20
Media del dataset completo: 50.17
Deviazione standard del dataset completo: 28.81
```

6-Generazione di un DataFrame con Distribuzione Ponderata

In [7]:

```
# Importa le librerie necessarie
import pandas as pd  # Per la manipolazione dei dati con DataFrame
import numpy as np   # Per la generazione di numeri casuali
import matplotlib.pyplot as plt  # Per la visualizzazione dei dati
from sklearn.model_selection import train_test_split  # Per la suddivisione del dataset

# Impostare il seed per la riproducibilità dei risultati
np.random.seed(42)

# Numero totale di elementi nel DataFrame
num_elementi = 1000

# Percentuale di "A"
percentuale_A = 0.7

# Generare la colonna con distribuzione desiderata
colonna = np.random.choice(['A', 'B'], size=num_elementi, p=[percentuale_A, 1 - percentuale_A])

# Creare il DataFrame
df = pd.DataFrame({'ColonnaAB': colonna})

# Restituire il DataFrame creato
df
```

Out[7]:

	ColonnaAB
0	A
1	B
2	B
3	A
4	A
...	...
995	A
996	B
997	A
998	B
999	A

1000 rows × 1 columns

7-Creazione di Tre Subset con Dimensioni Simili dal DataFrame

In [26]:

```
# Creare tre subset di dimensioni simili

# Estrai un sottoinsieme casuale del DataFrame df con una frazione del 1/3
subset1 = df.sample(frac=1/3)

# Rimuovi dal DataFrame df gli indici corrispondenti al subset1
df = df.drop(subset1.index)

# Estrai un sottoinsieme casuale del DataFrame df con una frazione del 1/2
subset2 = df.sample(frac=1/2)

# Rimuovi dal DataFrame df gli indici corrispondenti al subset2
df = df.drop(subset2.index)

# Il terzo subset (subset3) conterrà il resto dei dati nel DataFrame df
subset3 = df  # L'ultimo subset con il rimanente
df
```

Out[26]:

	Feature1	Feature2	Feature3	Feature4	Is_Outlier
8	2500000	50000	125	20005	True

8-Calcolo delle Percentuali di 'A' e 'B' nel Subset1

In [9]:

```
# Calcolare le percentuali di "A" e "B" per il subset1
percentuali_subset1 = subset1['ColonnaAB'].value_counts(normalize=True)
percentuali_subset1
```

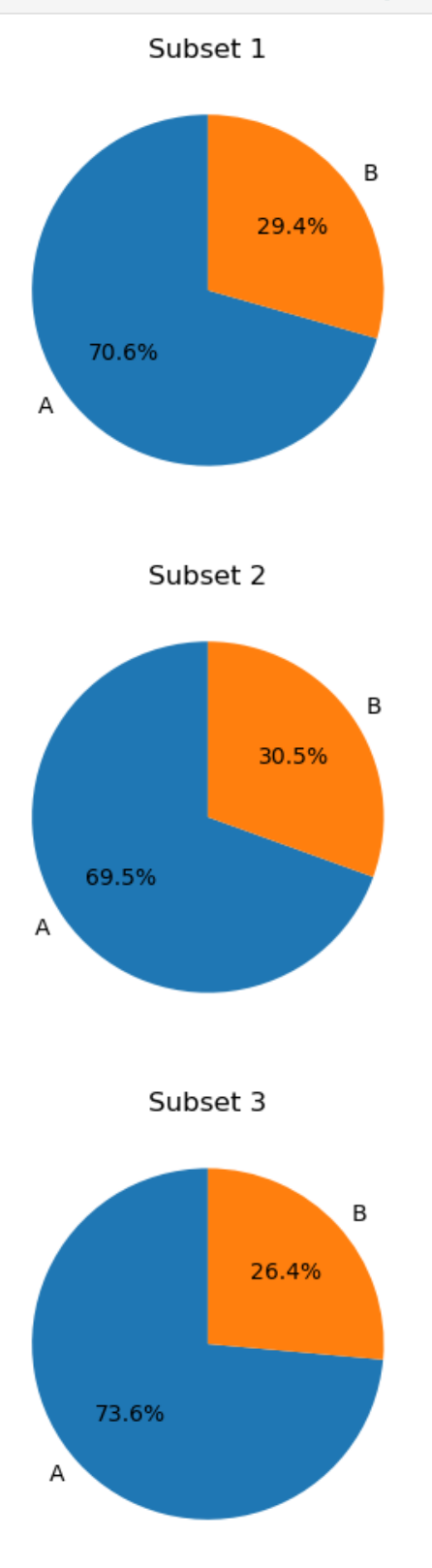
Out[9]:

```
A    0.705706
B    0.294294
Name: ColonnaAB, dtype: float64
```

9-Analisi delle Percentuali di 'A' e 'B' per Ogni Subset

In [10]:

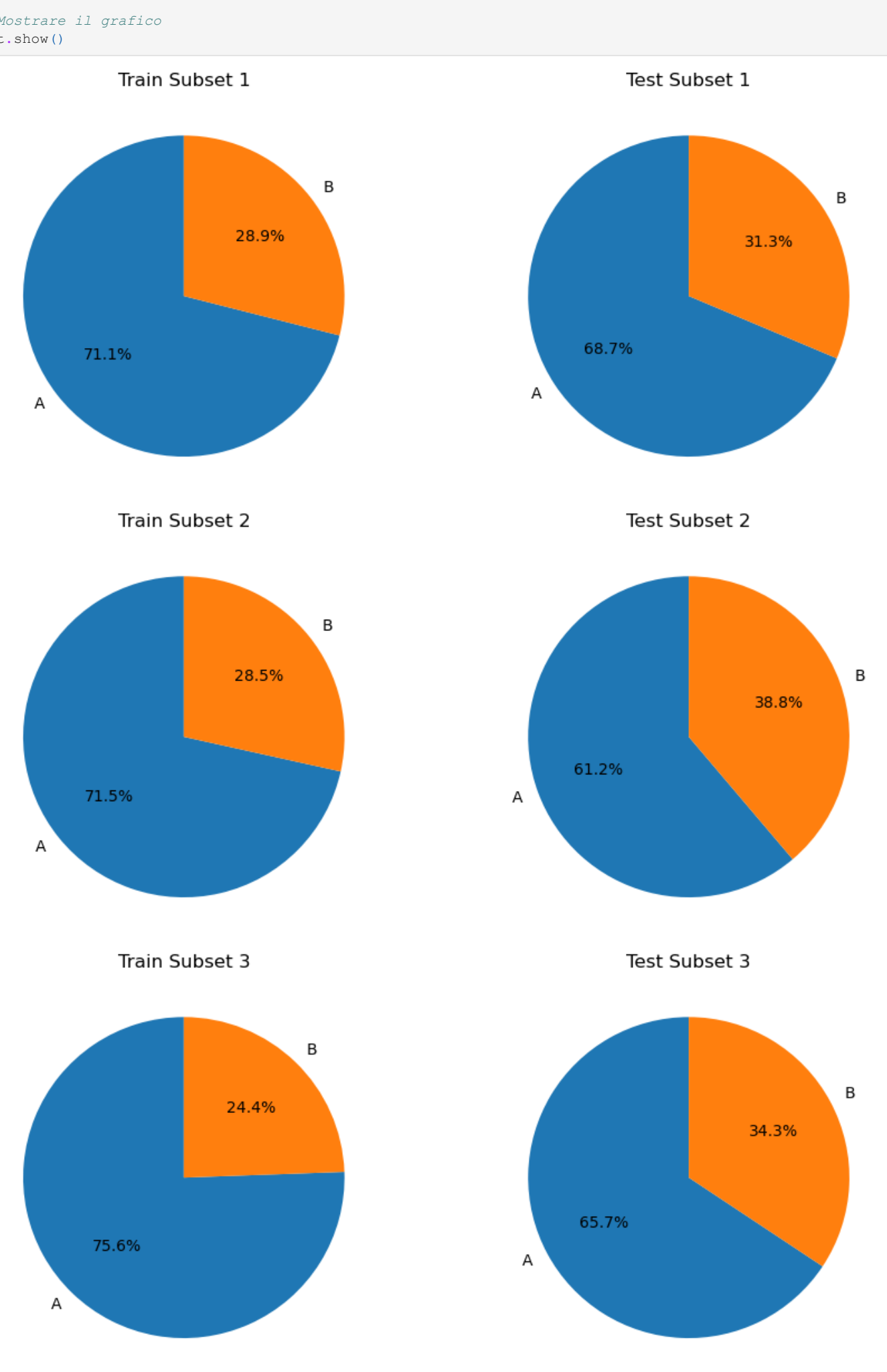
```
# Calcolare le percentuali di "A" e "B" per ogni subset
percentuali_subset1 = subset1['ColonnaAB'].value_counts(normalize=True)  # Calcola le percentuali per il subset1
percentuali_subset2 = subset2['ColonnaAB'].value_counts(normalize=True)  # Calcola le percentuali per il subset2
percentuali_subset3 = subset3['ColonnaAB'].value_counts(normalize=True)  # Calcola le percentuali per il subset3

# Creare i grafici a torta
fig, axs = plt.subplots(3, 1, figsize=(6, 12))  # Crea una griglia di subplot 3x1 con dimensioni della figura (6, 12)

# Plot per Subset 1
axs[0].pie(percentuali_subset1, labels=percentuali_subset1.index, autopct='%1.1f%%', startangle=90)  # Crea un grafico a torta per il Subset 1
axs[0].set_title('Subset 1')  # Imposta il titolo del subplot 1

# Plot per Subset 2
axs[1].pie(percentuali_subset2, labels=percentuali_subset2.index, autopct='%1.1f%%', startangle=90)  # Crea un grafico a torta per il Subset 2
axs[1].set_title('Subset 2')  # Imposta il titolo del subplot 2

# Plot per Subset 3
axs[2].pie(percentuali_subset3, labels=percentuali_subset3.index, autopct='%1.1f%%', startangle=90)  # Crea un grafico a torta per il Subset 3
axs[2].set_title('Subset 3')  # Imposta il titolo del subplot 3

# Mostrare il grafico
plt.show()  # Mostra tutti i subplot creati
```



10-Divisione dei Subset in Set di Addestramento e Test

In [11]:

```
# Dividere ciascun subset in training set e test set
train_subset1, test_subset1 = train_test_split(subset1, test_size=0.2, random_state=42)
train_subset2, test_subset2 = train_test_split(subset2, test_size=0.2, random_state=42)
train_subset3, test_subset3 = train_test_split(subset3, test_size=0.2, random_state=42)

# Creare il grafico con 6 torte
fig, axs = plt.subplots(3, 2, figsize=(10, 12))

# Definizione di una funzione per disegnare una torta con etichette
def draw_pie(ax, data, title):
    ax.pie(data, labels=data.index, autopct='%1.1f%%', startangle=90)
    ax.set_title(title)

# Prima riga di torte (Subset 1)
draw_pie(axs[0, 0], train_subset1['ColonnaAB'].value_counts(normalize=True), 'Train Subset 1')
draw_pie(axs[0, 1], test_subset1['ColonnaAB'].value_counts(normalize=True), 'Test Subset 1')

# Seconda riga di torte (Subset 2)
draw_pie(axs[1, 0], train_subset2['ColonnaAB'].value_counts(normalize=True), 'Train Subset 2')
draw_pie(axs[1, 1], test_subset2['ColonnaAB'].value_counts(normalize=True), 'Test Subset 2')

# Terza riga di torte (Subset 3)
draw_pie(axs[2, 0], train_subset3['ColonnaAB'].value_counts(normalize=True), 'Train Subset 3')
draw_pie(axs[2, 1], test_subset3['ColonnaAB'].value_counts(normalize=True), 'Test Subset 3')

# Regolare lo spaziamento tra i subplots per evitare sovrapposizioni
plt.tight_layout()

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



11-Analisi Statistica di un DataFrame

In [1]:

```
# Importa le librerie Pandas e Matplotlib.pyplot
import pandas as pd
import matplotlib.pyplot as plt

# Crea un DataFrame di esempio con una colonna chiamata "Valori"
data = {'Valori': [1, 2, 3, 4, 5, 10, 15, 20, 25, 300, 1000, 100000000, -50000000, -50]}
df = pd.DataFrame(data)

# Calcola la media dei valori nella colonna "Valori"
mean_value = df['Valori'].mean()

# Calcola la deviazione standard dei valori nella colonna "Valori"
std_dev = df['Valori'].std()

# Restituisce il valore della deviazione standard
std_dev
```

Out[1]:

```
30786384.39895254
```

12-Analisi Statistica dei Valori in un DataFrame

In [13]:

```
# Identifica gli outliers considerando ±3 sigma dalla media
outliers = df[(df['Valori'] > mean_value + 3 * std_dev) | (df['Valori'] < mean_value - 3 * std_dev)]
outliers
```

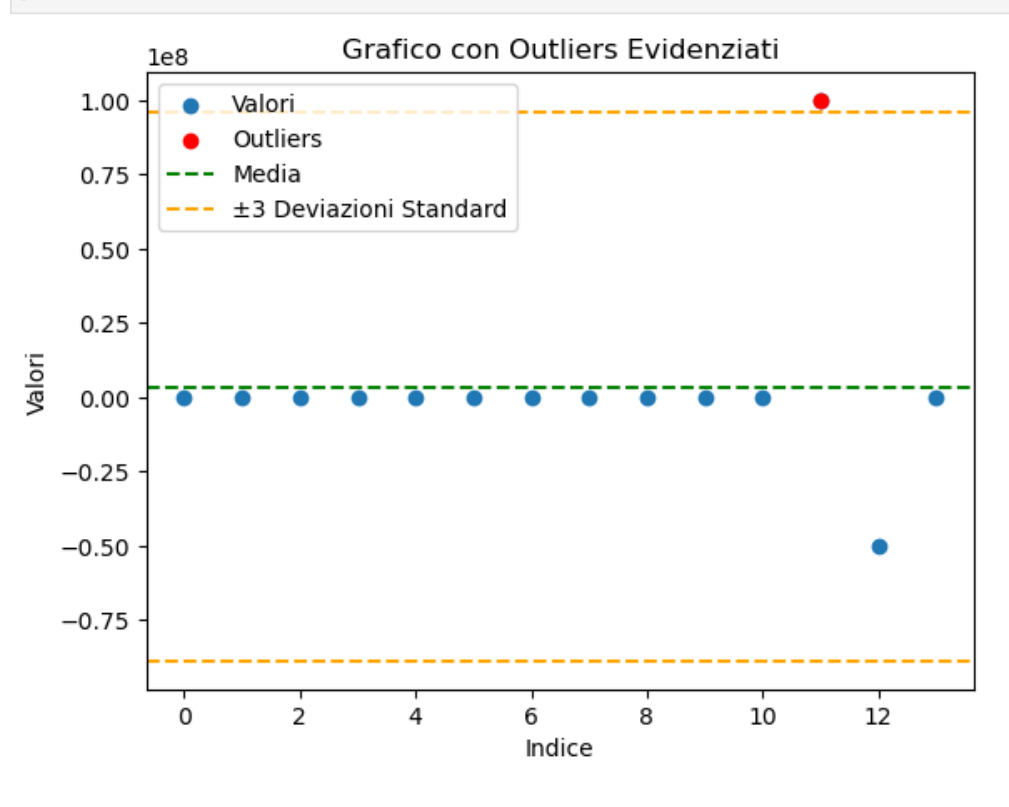
Out[13]:

	Valori
11	100000000

13-Grafico a Dispersione con Evidenziazione degli Outliers e Statistiche

In [14]:

```
# Crea un grafico a dispersione dei valori nel DataFrame df
plt.scatter(df.index, df['Valori'], label='Valori')

# Evidenzia gli outlier nel grafico con un colore rosso
plt.scatter(outliers.index, outliers['Valori'], color='red', label='Outliers')

# Aggiunge una linea orizzontale per rappresentare la media
plt.axhline(y=mean_value, color='green', linestyle='--', label='Media')

# Aggiunge linee orizzontali per rappresentare ±3 deviazioni standard dalla media
plt.axhline(y=mean_value + 3 * std_dev, color='orange', linestyle='--', label='±3 Deviazioni Standard')
plt.axhline(y=mean_value - 3 * std_dev, color='orange', linestyle='--')

# Aggiunge etichette e legenda al grafico
plt.xlabel('Indice')
plt.ylabel('Valori')
plt.title('Grafico con Outliers Evidenziati')
plt.legend()

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



14-Identificazione degli Outliers nelle Features del DataFrame

In [15]:

```
import pandas as pd  # Importa la libreria pandas per la manipolazione dei dati
import matplotlib.pyplot as plt  # Importa la libreria matplotlib per la visualizzazione dei dati

# Crea un DataFrame di esempio con 4 features
data = {'Feature1': [1, 2000, 3, 4, 50000, 10, 15, 20, 2500000, 300000000, 100000000],
        'Feature2': [2, 4, 6, 8, 10, 20, 30, 40, 50000, 60, 200],
        'Feature3': [5, 10, 15, 20000, 25, 50, 75, 100, 125, 150, 500000],
        'Feature4': [1, -20000000, 3, 4000000000, 5, 10, 15, 20, 20005, 30, 10000]}

df = pd.DataFrame(data)

# Definisci il numero minimo di features che devono superare la soglia per considerare un dato un outlier
min_features_threshold = 1
k = 2  # Fattore di moltiplicazione per l'intervallo di confidenza

# Lista per salvare gli indici degli outliers
outlier_indices = []

# Itera su ogni feature
for feature in df.columns:
    mean_value = df[feature].mean()
    std_dev = df[feature].std()

    # Identifica gli outliers per ciascuna feature e li salva come una nuova colonna nel DataFrame
    df['Outlier_' + feature] = (df[feature] > mean_value + k * std_dev) | (df[feature] < mean_value - k * std_dev)

# Visualizza il DataFrame con le colonne aggiuntive per gli outlier
df
```

Out[15]:

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1	Outlier_Feature2	Outlier_Feature3	Outlier_Feature4
0	1	2	5	1	False	False	False	False
1	2000	4	10	-20000000	False	False	False	False
2	3	6	15	3	False	False	False	False
3	4	8	20000	4000000000	False	False	False	True
4	50000	10	25	5	False	False	False	False
5	10	20	50	10	False	False	False	False
6	15	30	75	15	False	False	False	False
7	20	40	100	20	False	False	False	False
8	2500000	50000	125	20005	False	True	False	False
9	300000000	60	150	30	True	False	False	False
10	100000000	200	500000	10000	False	False	True	False

15-Calcolo del Numero Totale di Outliers per Ogni Riga nel DataFrame

In [16]:

```
# Calcola il numero totale di outlier per ogni riga nel DataFrame df
df['Num_Outliers'] = df.filter(like='Outlier_').sum(axis=1)
df
```

Out[16]:

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1	Outlier_Feature2	Outlier_Feature3	Outlier_Feature4	Num_Outliers
0	1	2	5	1	False	False	False	False	0
1	2000	4	10	-20000000	False	False	False	False	0
2	3	6	15	3	False	False	False	False	0
3	4	8	20000	4000000000	False	False	False	True	1
4	50000	10	25	5	False	False	False	False	0
5	10	20	50	10	False	False	False	False	0
6	15	30	75	15	False	False	False	False	0
7	20	40	100	20	False	False	False	False	0
8	2500000	50000	125	20005	False	True	False	False	1
9	300000000	60	150	30	True	False	False	False	1
10	100000000	200	500000	10000	False	False	True	False	1

16-Identificazione e Marcatura degli Outliers nel DataFrame

In [18]:

```
# Calcola il numero di features che superano la soglia per ogni riga
df['Num_Outliers'] = df.filter(like='Outlier_').sum(axis=1)
# Filtra i dati per mantenere solo le righe con almeno il numero minimo di features superanti la soglia
outliers = df[df['Num_Outliers'] >= min_features_threshold]
# Aggiungi una colonna che indica se il record è un outlier o meno
df['Is_Outlier'] = df.index.isin(outliers.index)
# Rimuovi colonne ausiliarie
df.drop(df.filter(like='Outlier_').columns, axis=1, inplace=True)
df.drop('Num_Outliers', axis=1, inplace=True)
df
```

Out[18]:

	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	2000	4	10	-20000000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	True
4	50000	10	25	5	False
5	10	20	50	10	False
6	15	30	75	15	False
7	20	40	100	20	False
8	2500000	50000	125	20005	True
9	300000000	60	150	30	True
10	100000000	200	500000	10000	True

17-Organizzazione dei Grafici in una Matrice

In [20]:

```
# Organizza i grafici in una matrice, con una colonna e 4 righe
num_features = len(df.columns) - 1  # Escludi la colonna 'Is_Outlier'
num_features #num_features viene assegnata al numero di colonne nel DataFrame meno uno, poiché stiamo escludendo la colonna 'Is_Outlier'
# se il DataFrame ha 5 colonne, di cui una è 'Is_Outlier', allora num_features sarà 4, poiché stiamo contando solo le colonne che rappresentano le feature effettive.
```

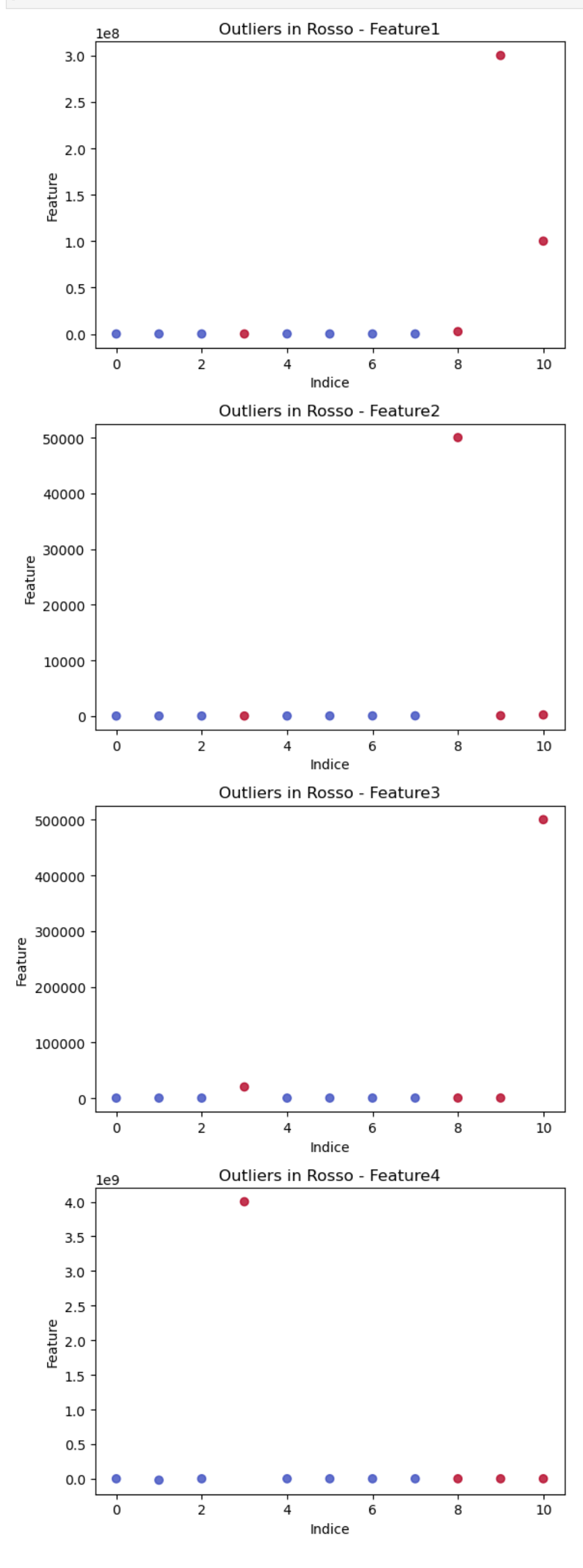
Out[20]:

```
4
```

18-Visualizzazione delle Features con Outliers Evidenziati

In [21]:

```
# Definizione del numero di righe e colonne per la visualizzazione dei dati
num_rows = num_features
num_cols = 1  # Una colonna
# Creazione di una nuova figura matplotlib con dimensioni specificate
plt.figure(figsize=(6, 4 * num_rows))
# Iterazione attraverso le colonne del DataFrame per visualizzare i dati
for i, feature in enumerate(df.columns[:-1]):  # Escludi la colonna 'Is_Outlier'
    # Creazione di un sotto-grafico nella posizione specificata
    plt.subplot(num_rows, num_cols, i + 1)
    # Tracciamento di uno scatter plot dei dati
    plt.scatter(df.index, df[feature], c=df['Is_Outlier'], cmap='coolwarm', alpha=0.8)
    # Impostazione del titolo del sotto-grafico
    plt.title(f'Outliers in Rosso - {feature}')
    # Etichettatura dell'asse x
    plt.xlabel('Indice')
    # Etichettatura dell'asse y
    plt.ylabel('Feature')
# Ottimizzazione del layout dei sotto-grafici
plt.tight_layout()
# Visualizzazione della figura completa
plt.show()
```



19-Eliminazione delle Righe Corrispondenti agli Outliers

In [22]:

```
# Elimina le righe corrispondenti agli outliers quelli che hanno almeno una features fuoriscala

# Filtraggio del DataFrame df per mantenere solo le righe dove il valore nella colonna 'Is_Outlier' è False
df_filtered = df[df['Is_Outlier'] == False]

# Visualizzazione del DataFrame filtrato
df_filtered
```

Out[22]:

	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	2000	4	10	-20000000	False
2	3	6	15	3	False
4	50000	10	25	5	False
5	10	20	50	10	False
6	15	30	75	15	False
7	20	40	100	20	False

20-Calcolo della Deviazione Standard

In [24]:

```
def calcola_deviazione_standard(lista):
    # Calcola il numero di elementi nella lista
    n = len(lista)
    # Calcola la media della lista
    media = sum(lista) / n
    # Calcola la somma dei quadrati delle differenze dalla media
    somma_quadrati_diff = sum((x - media) ** 2 for x in lista)
    # Calcola la deviazione standard
    deviazione_standard = (somma_quadrati_diff / n) ** 0.5
    return deviazione_standard
# Esempio di utilizzo
numero_lista = [1, 2, 3, 4, 50]
deviazione_standard = calcola_deviazione_standard(numero_lista)
# Stampa il risultato
print(f"La deviazione standard della lista è: {deviazione_standard}")
```

```
La deviazione standard della lista è: 19.026297590440446
```