Outliers

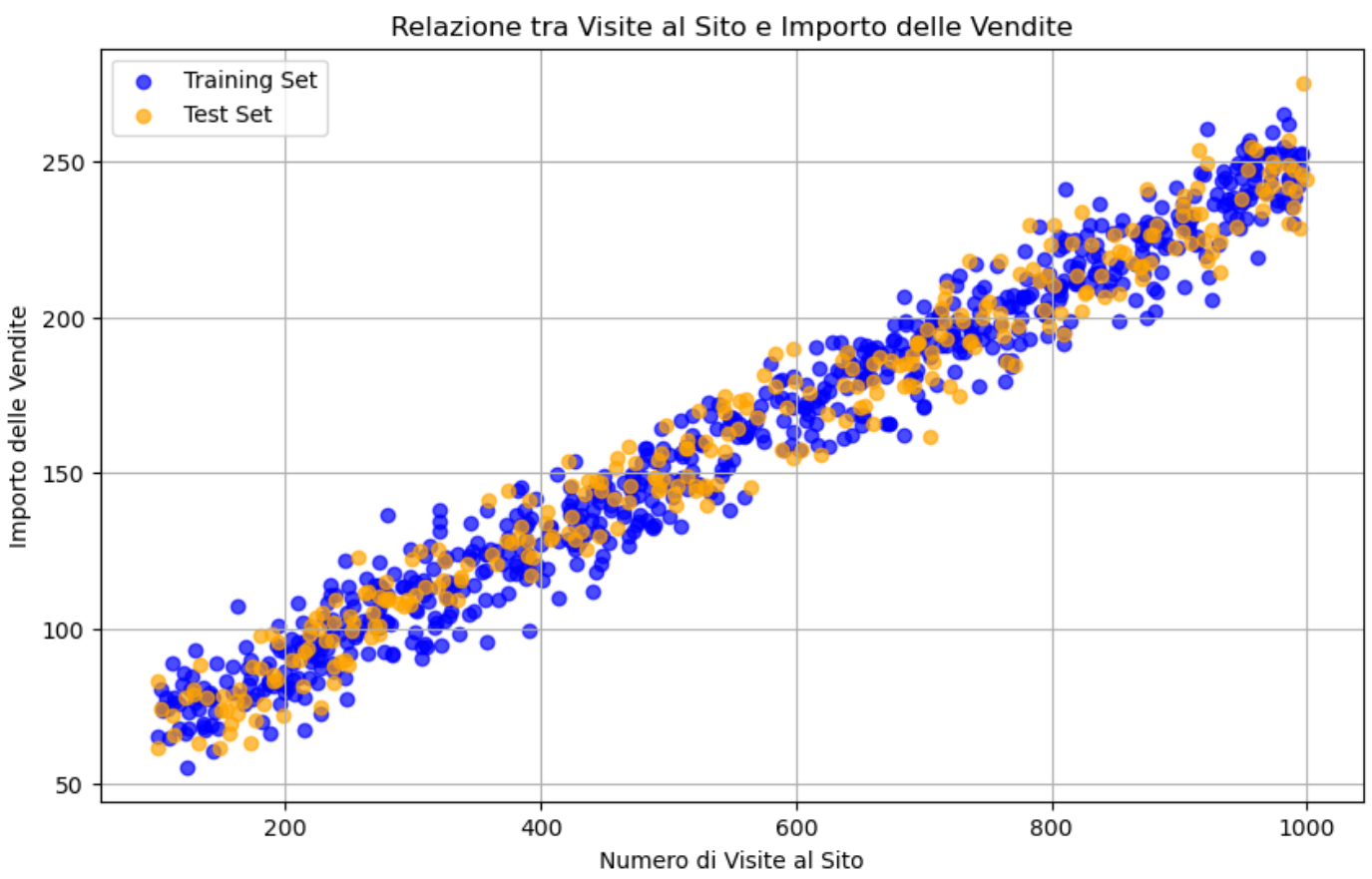
1.0 Generazione di Dati e Divisione in Training e Test Set per Predizione di Pesi da Altezze

In [4]:
```
import numpy as np
from sklearn.model_selection import train_test_split
np.random.seed(0)
altezze = np.random.normal(0, 5, 100)
pesi = 0.5 * altezze + np.random.normal(0, 5, 100)
X_train, X_test, y_train, y_test = train_test_split(altezze, pesi, test_size=0.3, random
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,)
```

1.1 Analisi della Relazione tra Visite al Sito e Importo delle Vendite: Generazione di Dati Casuali e Suddivisione in Training e Test Set

In [5]:
```
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, tes

# 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.s
print("Dimensioni del Test Set (visite al sito e importo delle vendite):", X_test.shape,
```




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

1.2 Analisi della Distribuzione di Classi in Training e Test Set: Generazione di Dati Casuali e Suddivisione Equa

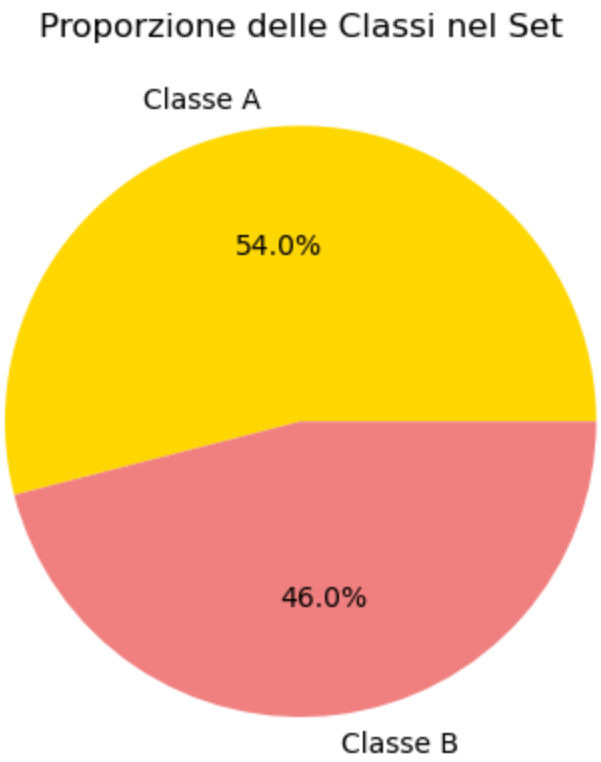
In [7]:

```python
from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(1)
X = np.random.rand(100, 2)
y = np.random.choice(['A', 'B'], size=100)

proporzione_classe_A = sum(y == 'A') / len(y)
proporzione_classe_B = 1 - proporzione_classe_A

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

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

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
```

1.3 Visualizzazione della Proporzione delle Classi nel Set attraverso un Diagramma a Torta

In [8]:
```
labels = ['Classe A', 'Classe B']
colors = ['gold', 'lightcoral']
plt.pie([proporzione_classe_A, proporzione_classe_B], labels=labels, colors=colors, auto
plt.title('Proporzione delle Classi nel Set')
plt.show()
```



1.4 Analisi Statistica: Campione Casuale vs Dataset Completo

In [10]:
```
import random
import numpy as np

dataset=[]
# Creazione di un dataset di 1000 elementi (ad esempio, dati casuali)
for i in range(1000):
    dataset.append(random.randint(1, 100))

# Estrazione di un campione casuale semplice di 50 elementi dal dataset
campione_casuale = random.sample(dataset, 300)

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

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

```
deviazione_standard_dataset = np.std(dataset)

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

```
Media del campione casuale:  51.32
Deviazione standard del campione casuale:  28.66
Media del dataset completo:  51.23
Deviazione standard del dataset completo:  28.88
```

1.5 Generazione di DataFrame con Distribuzione Controllata: Colonna 'ColonnaAB

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Impostare il seed per la riproducibilità
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 - percentu

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

Out[11]:

	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

1.6 Creazione di Subset da un DataFrame: Partizione Equa in Tre Gruppi

In [12]:
```
# Creare tre subset di dimensioni simili
sample(frac=1/3)
```

```
df = df.drop(subset1.index)

subset2 = df.sample(frac=1/2)
df = df.drop(subset2.index)

subset3 = df  # L'ultimo subset con il rimanente
```

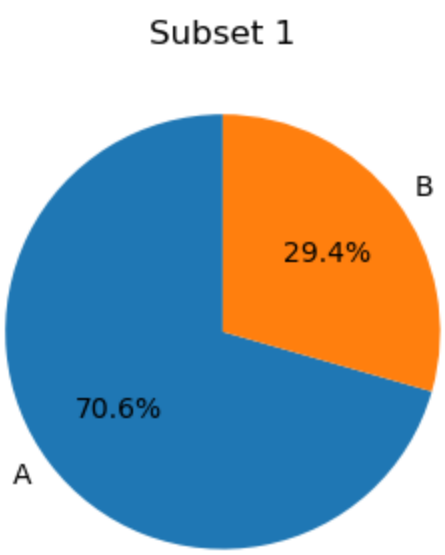
1.7 Analisi delle Percentuali di 'ColonnaAB' nel Subset1

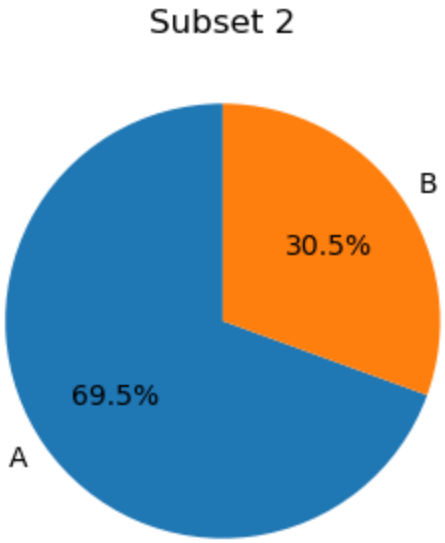
In [13]:
```
percentuali_subset1 = subset1['ColonnaAB'].value_counts(normalize=True)
percentuali_subset1
```

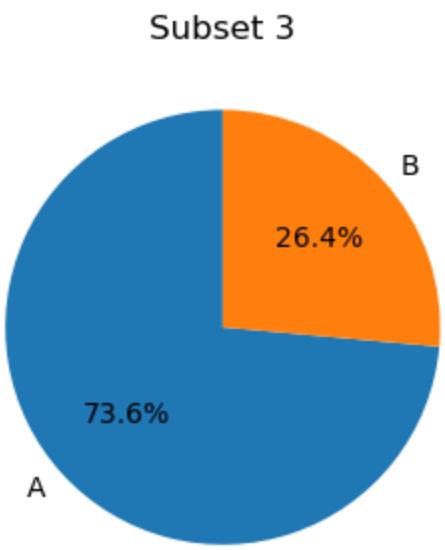
Out[13]:
```
ColonnaAB
A    0.705706
B    0.294294
Name: proportion, dtype: float64
```

1.8 Visualizzazione delle Percentuali di 'A' e 'B' nei Subset

In [14]:
```
# Calcolare le percentuali di "A" e "B" per ogni subset
percentuali_subset1 = subset1['ColonnaAB'].value_counts(normalize=True)
percentuali_subset2 = subset2['ColonnaAB'].value_counts(normalize=True)
percentuali_subset3 = subset3['ColonnaAB'].value_counts(normalize=True)

# Creare i grafici a torta
fig, axs = plt.subplots(3, 1, figsize=(6, 12))

# Subset 1
axs[0].pie(percentuali_subset1, labels=percentuali_subset1.index, autopct='%1.1f%%', sta
axs[0].set_title('Subset 1')

# Subset 2
axs[1].pie(percentuali_subset2, labels=percentuali_subset2.index, autopct='%1.1f%%', sta
axs[1].set_title('Subset 2')

# Subset 3
axs[2].pie(percentuali_subset3, labels=percentuali_subset3.index, autopct='%1.1f%%', sta
axs[2].set_title('Subset 3')

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





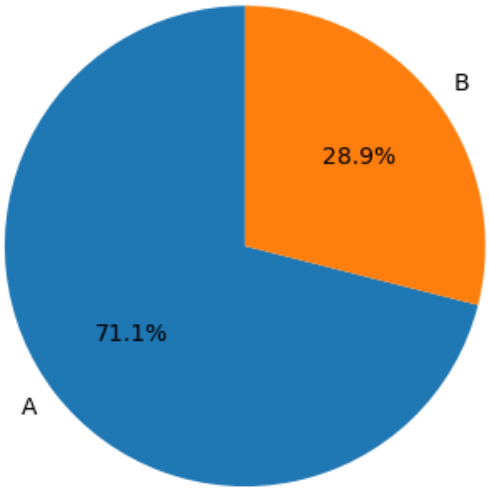


1.9 Divisione dei Subset in Training e Test Sets con Analisi Percentuale di 'A' e 'B'

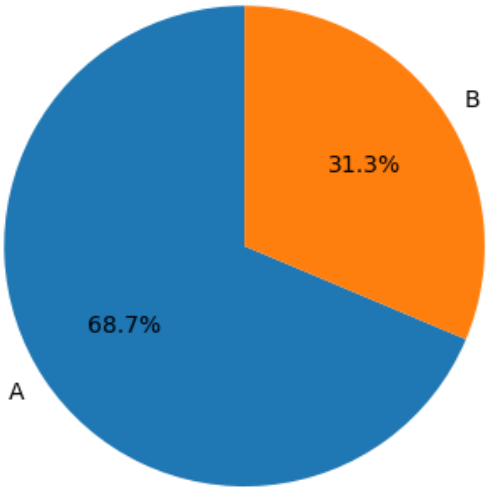
In [15]:

```
# 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))

# 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 Subs
draw_pie(axs[0, 1], test_subset1['ColonnaAB'].value_counts(normalize=True), 'Test Subset

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

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

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

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

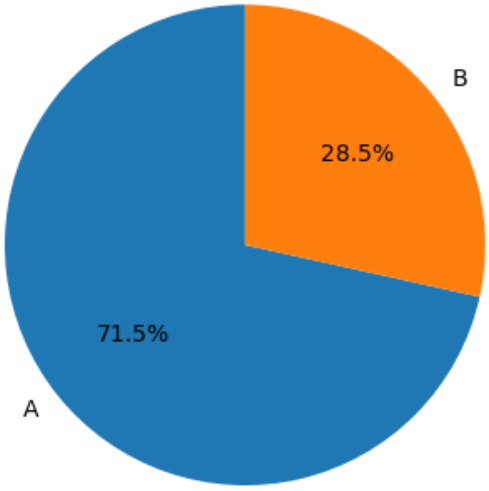
Train Subset 1



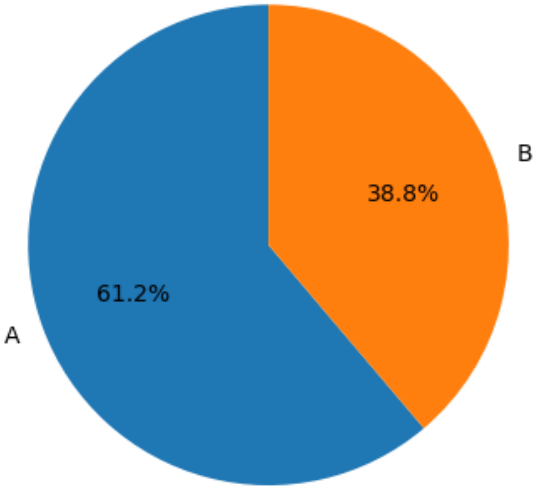
Test Subset 1



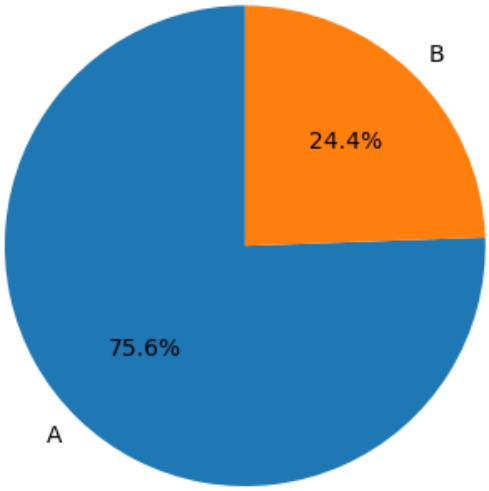
Train Subset 2



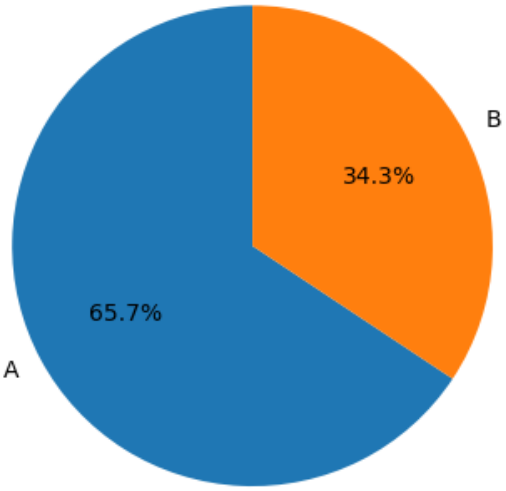
Test Subset 2



Train Subset 3



Test Subset 3



2.0 Analisi di Valori: Calcolo della Media e Deviazione Standard

In [16]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Crea un DataFrame di esempio
data = {'Valori': [1, 2, 3, 4, 5, 10, 15, 20, 25, 300, 1000, 100000000, -50000000, -50]}
df = pd.DataFrame(data)
# Lista con outliers da entrambi i lati

# Calcola la media e la deviazione standard
mean_value = df['Valori'].mean()
std_dev = df['Valori'].std()
std_dev
```

Out[16]:
```
30786384.39895254
```

2.1 Rilevamento degli Outliers con Limite a ±3 Deviazioni Standard dalla Media

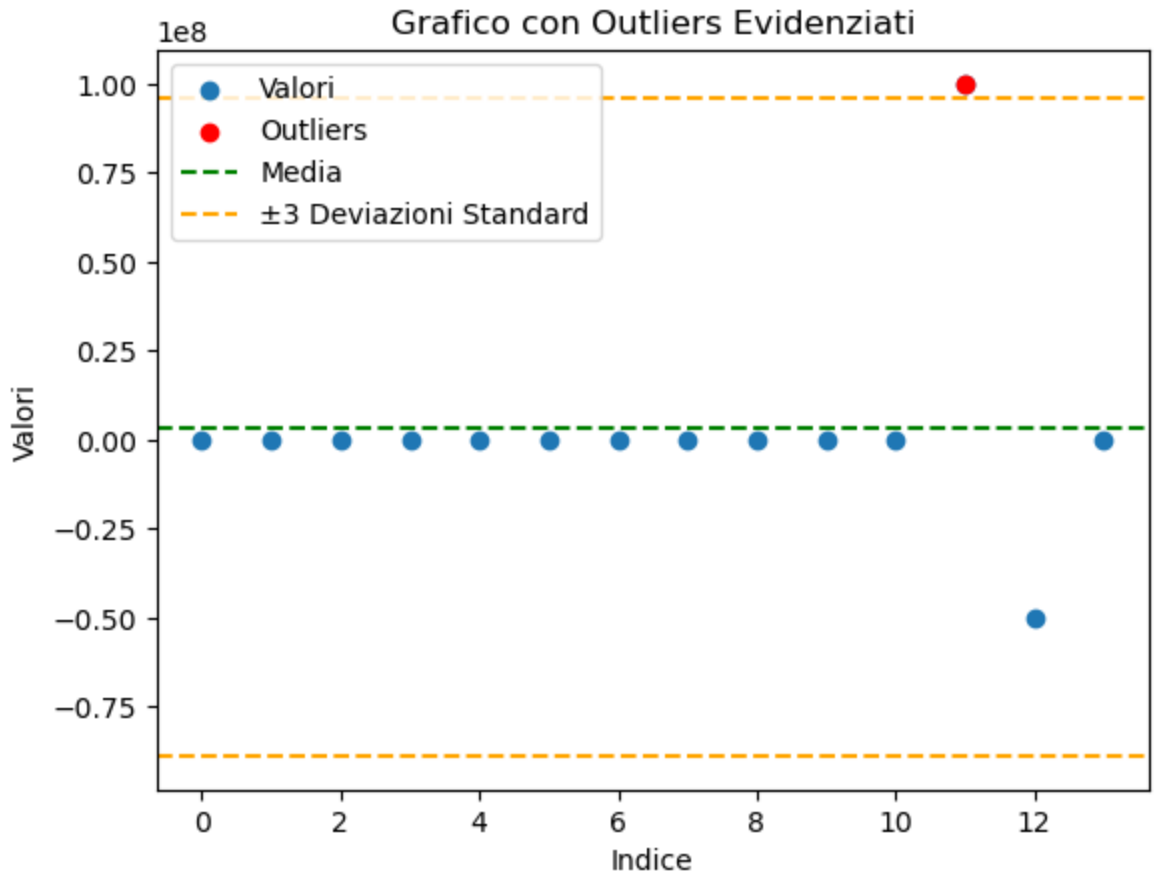
In [17]:
```
# Identifica gli outliers considerando ±3 sigma dalla media
outliers = df[(df['Valori'] > mean_value + 3 * std_dev) | (df['Valori'] < mean_value - 3
outliers
```

Out[17]:

	Valori
11	100000000

2.2 Visualizzazione dei Dati con Evidenziazione degli Outliers e Statistiche Aggiuntive

In [18]:
```
# Crea un grafico a dispersione
plt.scatter(df.index, df['Valori'], label='Valori')

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

# Aggiungi la media e la deviazione standard al grafico
plt.axhline(y=mean_value, color='green', linestyle='--', label='Media')
plt.axhline(y=mean_value + 3 * std_dev, color='orange', linestyle='--', label='±3 Deviaz
plt.axhline(y=mean_value - 3 * std_dev, color='orange', linestyle='--')

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

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



2.3 Identificazione degli Outliers in un DataFrame Multivariato con Soglia Minima di Features

In [19]:

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

# 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 u
min_features_threshold = 1
k=2 #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
    df['Outlier_' + feature] = (df[feature] > mean_value + k * std_dev) | (df[feature] <
df
```

Out[19]:

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

2.4 Conteggio degli Outliers per Ciascuna Riga nel DataFrame

In [20]:
```
df['Num_Outliers'] = df.filter(like='Outlier_').sum(axis=1)
df
```

Out[20]:

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

2.5 Identificazione e Filtraggio di Outliers basati sul Numero Minimo di Features che Superano la Soglia

In [23]:
```
# 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 supe
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)

onne ausiliarie
```

```
df.drop(df.filter(like='Outlier_').columns, axis=1, inplace=True)
df.drop('Num_Outliers', axis=1, inplace=True)
df
```

Out[23]:

	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

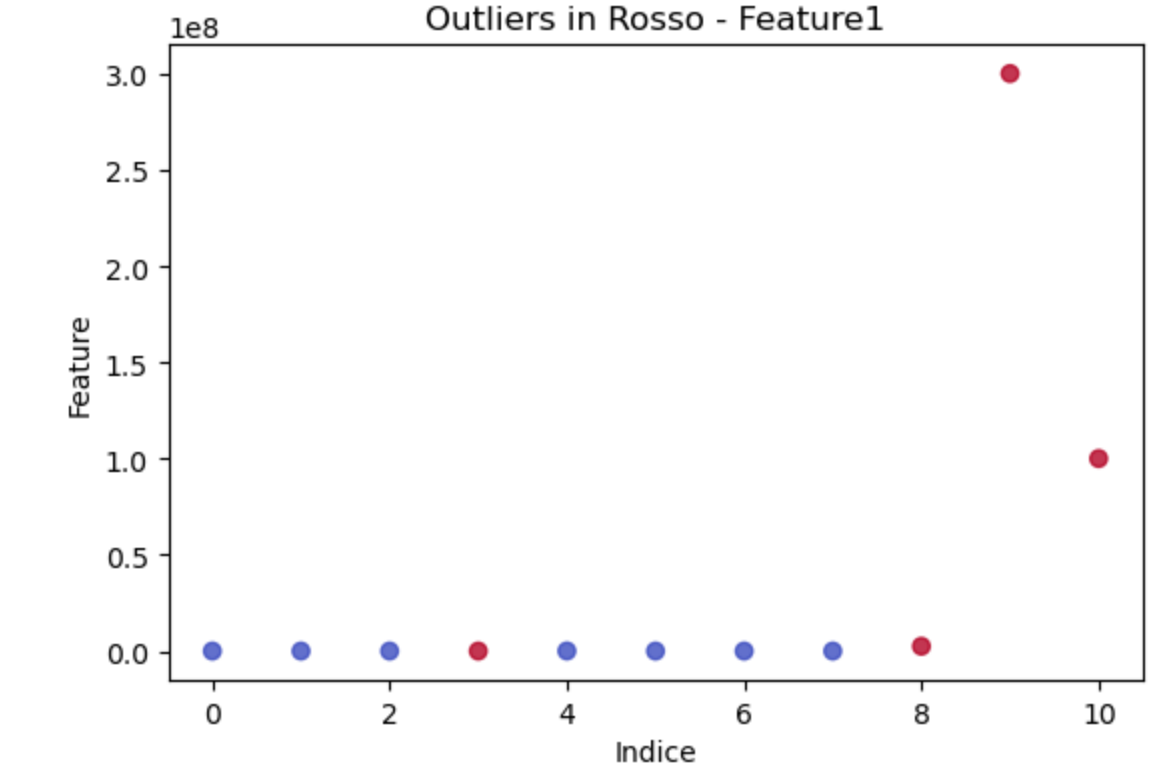
2.6 Organizzazione dei Grafici in una Matrice con 4 Righe e 1 Colonna per ciascuna Feature

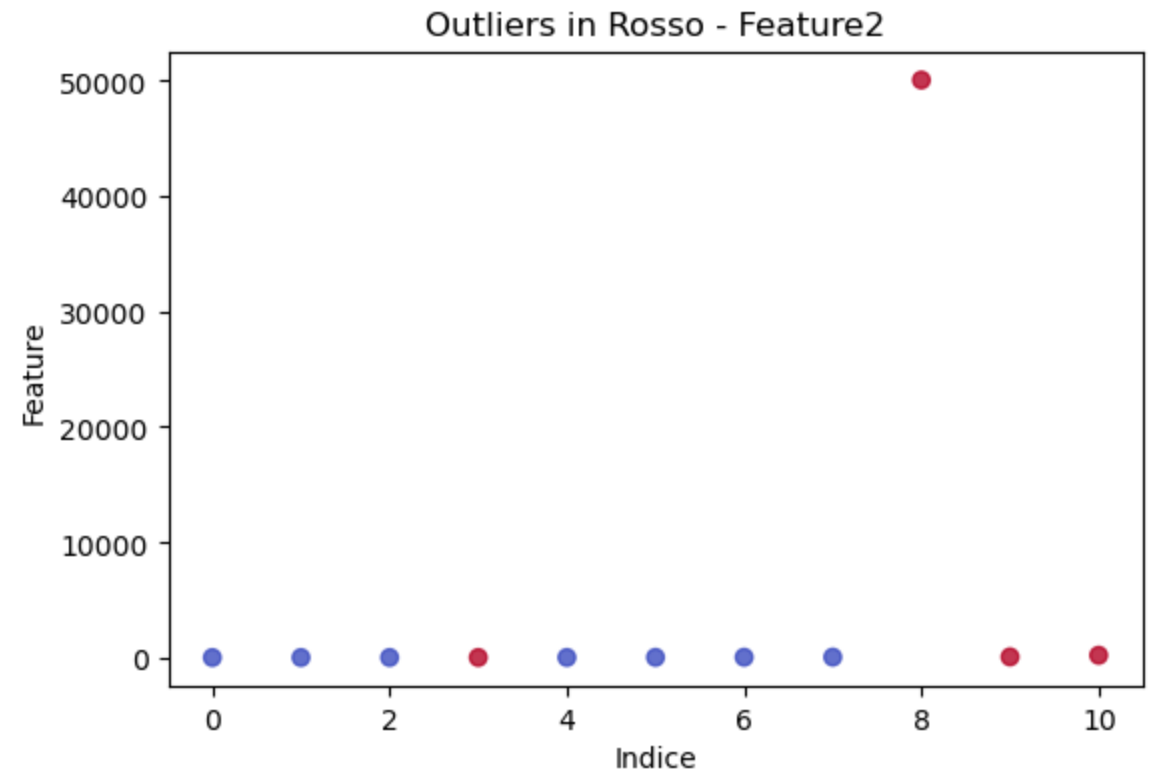
In [26]:
```
# Organizza i grafici in una matrice, con una colonna e 4 righe
num_features = len(df.columns) - 1  # Escludi la colonna 'Is_Outlier'
num_features
```

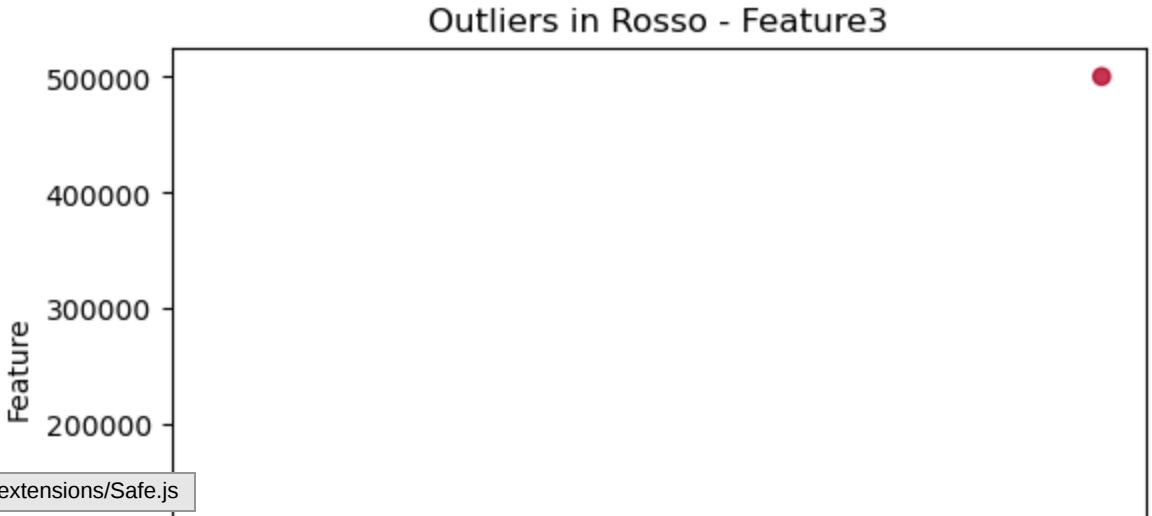
Out[26]: 4

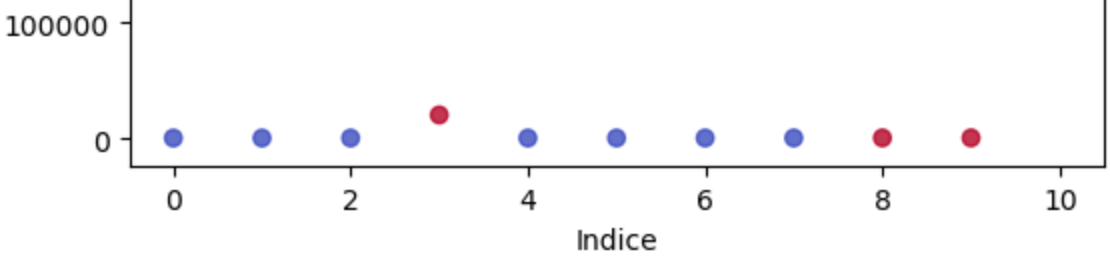
2.7 Visualizzazione delle Features con Evidenziazione degli Outliers

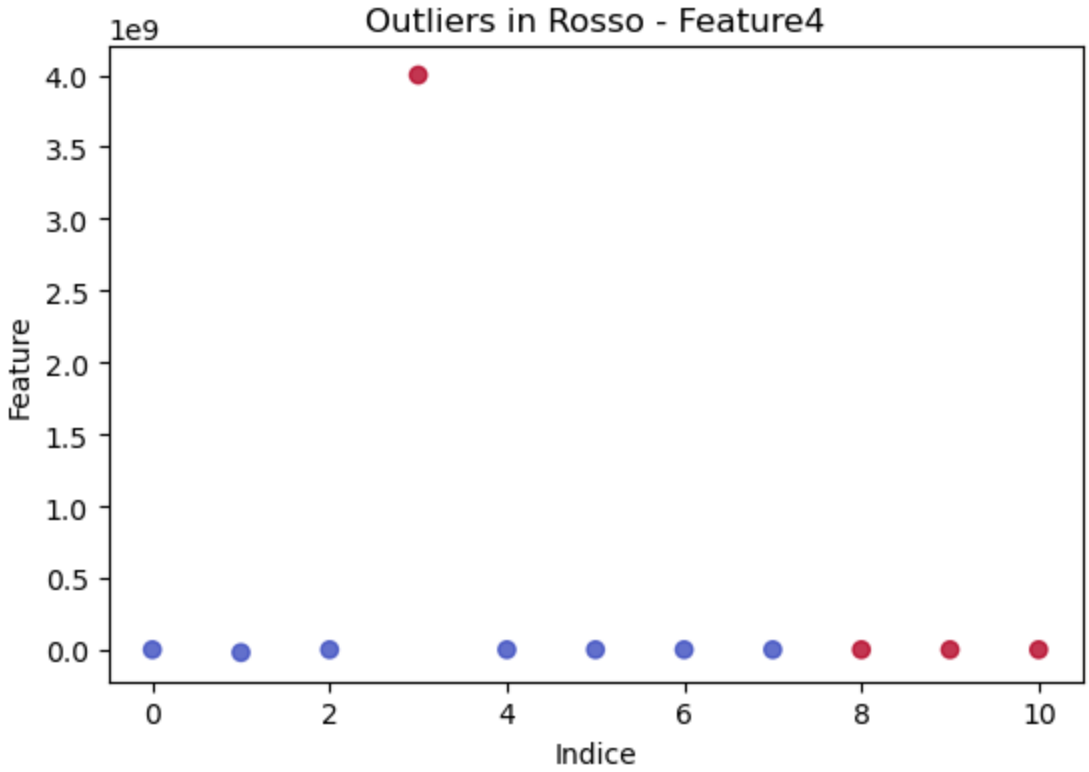
In [27]:
```
num_rows = num_features
num_cols = 1  # Una colonna

plt.figure(figsize=(6, 4 * num_rows))
for i, feature in enumerate(df.columns[:-1]):  # Escludi la colonna 'Is_Outlier'
    plt.subplot(num_rows, num_cols, i + 1)
    plt.scatter(df.index, df[feature], c=df['Is_Outlier'], cmap='coolwarm', alpha=0.8)
    plt.title(f'Outliers in Rosso - {feature}')
    plt.xlabel('Indice')
    plt.ylabel('Feature')

plt.tight_layout()
plt.show()
```











2.8 Rimozione delle Righe Contenenti Outliers o con almeno una Feature Fuoriscala

In [28]:
```
# Elimina le righe corrispondenti agli outliers quelli che hanno almeno una features fuo
df_filtered = df[df['Is_Outlier'] == False]
df_filtered
```

Out[28]:

	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

2.9 Calcolo Manuale della Deviazione Standard per una Lista di Numeri

In [29]:
```
def calcola_deviazione_standard(lista):
    # 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
```

In [ ]: