

# LAVORO DI GRUPPO OUTLIERS & SPLITTING DATASET

March 12, 2024

## 1 OUTLIERS & SPLITTING DATASET

1.1 CREATO DA GRANIERI JOELE, TASSONE LEONARDO, RAPHAEL RODRIGO E RADISHA WARNAKULASURIYA

### 1.2 SPLITTING DATASET

#### 1.2.1 Analisi e Preparazione dei Dati: Un Esempio con Altezze e Pesi

```
[1]: 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,)

Questo codice esegue le seguenti operazioni:

Genera dati casuali per altezze e pesi: Crea un set di dati di 100 punti, dove le altezze sono generate casualmente da una distribuzione normale con media 160 e deviazione standard 10. I pesi sono generati come il 50% delle altezze più un rumore casuale da una distribuzione normale con media 0 e deviazione standard 5.

Suddivide il dataset in training set e test set: Il dataset viene suddiviso in un training set (70% dei dati) e un test set (30% dei dati). Questo è fatto per avere un set di dati separato (test set) per valutare le prestazioni del modello che sarà addestrato sul training set.

Stampa le dimensioni dei training set e test set: Infine, il codice stampa le dimensioni dei training set e test set. Questo è utile per verificare che la suddivisione dei dati sia stata eseguita correttamente.

In sintesi, questo codice è un esempio di come si potrebbe preparare un set di dati per un problema di apprendimento supervisionato, come la regressione lineare, dove si cerca di prevedere il peso (variabile dipendente) in base all'altezza (variabile indipendente).

### 1.2.2 Analisi Statistica di un Campione Estratto da un Dataset: Un Esempio con Numeri Casuali

```
[2]: 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
media_dataset = np.mean(dataset)
deviazione_standard_dataset = np.std(dataset)

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

```
Media del campione casuale:  51.10
Deviazione standard del campione casuale:  28.55
Media del dataset completo:  51.26
Deviazione standard del dataset completo:  28.98
```

Questo codice esegue le seguenti operazioni:

Crea un dataset di 1000 elementi: Genera un dataset di 1000 numeri interi casuali tra 1 e 100.

Estrae un campione casuale dal dataset: Estrae un campione casuale di 300 elementi dal dataset.

Calcola la media e la deviazione standard del campione: Calcola la media e la deviazione standard dei valori nel campione casuale.

Calcola la media e la deviazione standard del dataset completo: Calcola la media e la deviazione standard dei valori nel dataset completo.

Stampa i risultati: Stampa la media e la deviazione standard sia del campione casuale che del dataset completo.

In sintesi, questo codice è un esempio di come si potrebbe eseguire un'analisi statistica su un campione di dati estratto da un dataset più grande. Questo è un concetto comune in statistica, dove spesso si lavora con campioni di dati piuttosto che con l'intero dataset. Questo può essere utile quando il dataset completo è troppo grande per essere gestito efficacemente, o quando si desidera stimare le proprietà di una popolazione più grande basandosi su un campione di essa.

### 1.2.3 Generazione e Analisi di un DataFrame con Distribuzione Specifica di Categorie

```
[3]: 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 -
↪percentuale_A])

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

# Calcolare la frequenza delle diverse categorie
frequenze = df['ColonnaAB'].value_counts(normalize=True)

print(f"Frequenze relative delle categorie:\n{frequenze}")
```

Frequenze relative delle categorie:

A 0.712

B 0.288

Name: ColonnaAB, dtype: float64

Questo codice esegue le seguenti operazioni:

Imposta il seed per la riproducibilità: Questo garantisce che i risultati siano riproducibili ogni volta che il codice viene eseguito.

Genera una colonna di dati: Crea una colonna di 1000 elementi, ognuno dei quali è 'A' o 'B'. La probabilità che un elemento sia 'A' è del 70% (come specificato dalla variabile `percentuale_A`), e la probabilità che un elemento sia 'B' è del 30% (che è  $1 - \text{percentuale\_A}$ ).

Crea un DataFrame: Crea un DataFrame di pandas con la colonna generata.

Calcola la frequenza delle categorie: Calcola la frequenza relativa di 'A' e 'B' nel DataFrame, cioè la percentuale di 'A' e 'B' nel DataFrame.

Stampa le frequenze relative: Infine, stampa le frequenze relative delle categorie 'A' e 'B'.

In sintesi, questo codice è un esempio di come si potrebbe generare un DataFrame con una distribuzione specifica di categorie e calcolare le loro frequenze relative. Questo può essere utile in molte applicazioni, come la simulazione di dati o il test di metodi statistici.

#### 1.2.4 Analisi Statistica e Visualizzazione di Subset di un DataFrame: Un Esempio con Grafici a Torta

```
[4]: # Dividere il DataFrame in tre subset
subset1 = df.sample(frac=0.33, random_state=42)
subset2 = df.drop(subset1.index).sample(frac=0.5, random_state=42)
subset3 = df.drop(subset1.index).drop(subset2.index)

# 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))

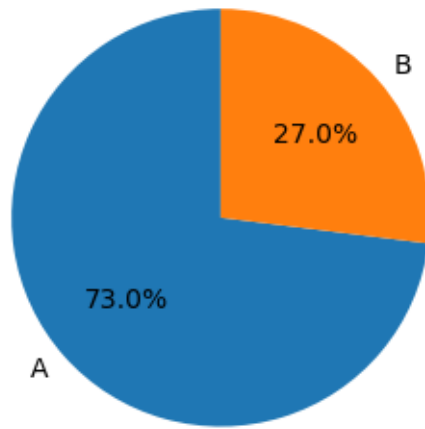
# Subset 1
axs[0].pie(percentuali_subset1, labels=percentuali_subset1.index, autopct='%1.1f%%', startangle=90)
axs[0].set_title('Subset 1')

# Subset 2
axs[1].pie(percentuali_subset2, labels=percentuali_subset2.index, autopct='%1.1f%%', startangle=90)
axs[1].set_title('Subset 2')

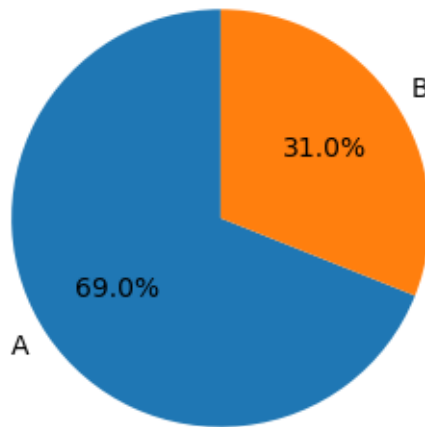
# Subset 3
axs[2].pie(percentuali_subset3, labels=percentuali_subset3.index, autopct='%1.1f%%', startangle=90)
axs[2].set_title('Subset 3')
```

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

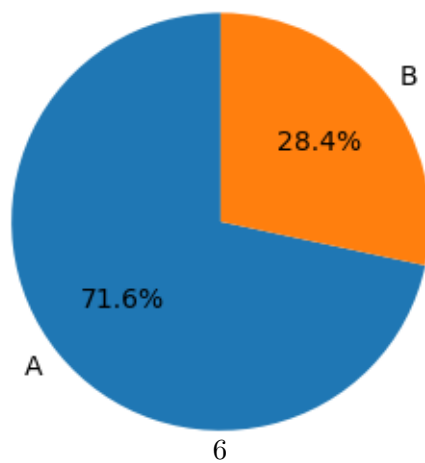
Subset 1



Subset 2



Subset 3



Questo codice esegue le seguenti operazioni:

Divide il DataFrame in tre subset: Il DataFrame viene suddiviso in tre subset. Il primo subset è un campione casuale del 33% del DataFrame originale. Il secondo subset è un campione casuale del 50% dei dati rimanenti. Il terzo subset contiene tutti i dati rimanenti.

Calcola le percentuali di 'A' e 'B' per ogni subset: Per ogni subset, calcola la percentuale di 'A' e 'B'.

Crea i grafici a torta: Per ogni subset, crea un grafico a torta che mostra la percentuale di 'A' e 'B'.

Mostra il grafico: Infine, mostra i grafici a torta.

In sintesi, questo codice è un esempio di come si potrebbe eseguire un'analisi statistica su diversi subset di un DataFrame e visualizzare i risultati con dei grafici a torta. Questo può essere utile per confrontare la distribuzione di categorie in diversi subset di un dataset.

### 1.2.5 Analisi Statistica e Visualizzazione di Training Set e Test Set: Un Esempio con Grafici a Torta

```
[5]: # 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)

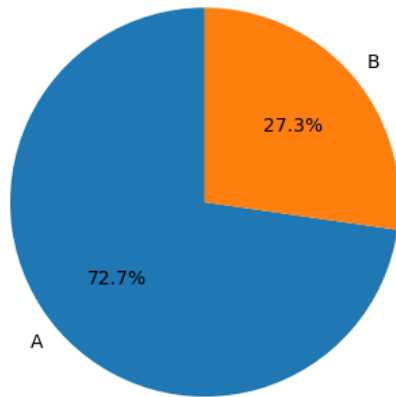
# Prima riga di torte (Subset 1)
draw_pie(axs[0, 0], train_subset1['ColonnaAB'].value_counts(normalize=True),
↳ 'Train Subset 1')
draw_pie(axs[0, 1], test_subset1['ColonnaAB'].value_counts(normalize=True),
↳ 'Test Subset 1')

# Seconda riga di torte (Subset 2)
```

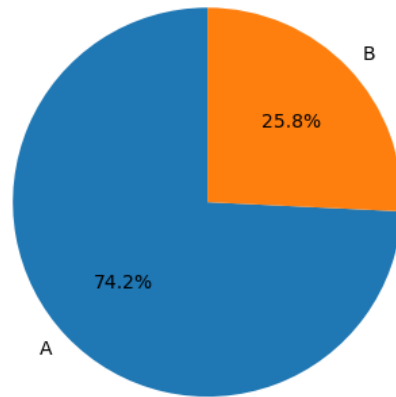
```
draw_pie(axes[1, 0], train_subset2['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Train Subset 2')  
draw_pie(axes[1, 1], test_subset2['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Test Subset 2')  
  
# Terza riga di torte (Subset 3)  
draw_pie(axes[2, 0], train_subset3['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Train Subset 3')  
draw_pie(axes[2, 1], test_subset3['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Test Subset 3')  
  
# Regolare lo spaziamento tra i subplots  
plt.tight_layout()  
  
# Mostrare il grafico  
plt.show()
```



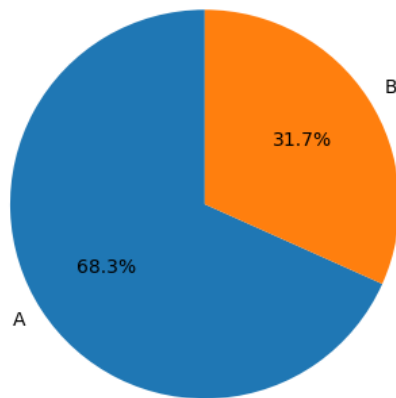
Train Subset 1



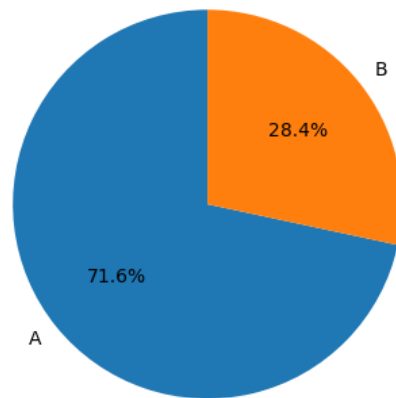
Test Subset 1



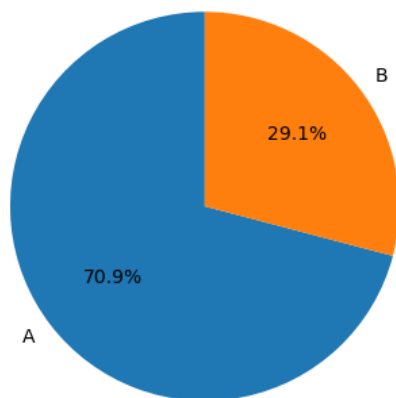
Train Subset 2



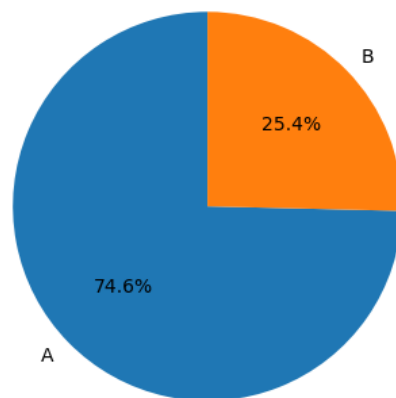
Test Subset 2



Train Subset 3



Test Subset 3



Questo codice esegue le seguenti operazioni:

Divide ciascun subset in training set e test set: Ogni subset viene suddiviso in un training set (80% dei dati) e un test set (20% dei dati). Questo è fatto per avere un set di dati separato (test set) per valutare le prestazioni del modello che sarà addestrato sul training set.

Crea un grafico con 6 torte: Per ogni training set e test set, crea un grafico a torta che mostra la percentuale di 'A' e 'B'.

Mostra il grafico: Infine, mostra i grafici a torta.

In sintesi, questo codice è un esempio di come si potrebbe eseguire un'analisi statistica su diversi training set e test set estratti da diversi subset di un DataFrame e visualizzare i risultati con dei grafici a torta. Questo può essere utile per confrontare la distribuzione di categorie in diversi training set e test set di un dataset.

### 1.2.6 Visualizzazione e Analisi di Training Set e Test Set: Un Esempio con Grafici a Torta

```
[6]: # 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)

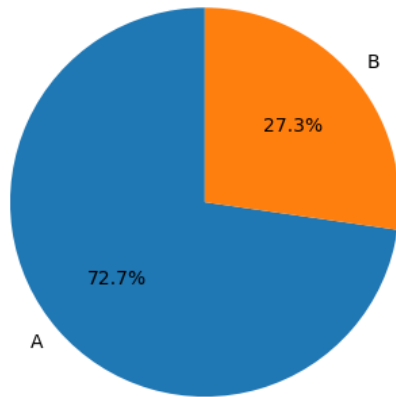
# Prima riga di torte (Subset 1)
draw_pie(axs[0, 0], train_subset1['ColonnaAB'].value_counts(normalize=True),
    ↪'Train Subset 1')
draw_pie(axs[0, 1], test_subset1['ColonnaAB'].value_counts(normalize=True),
    ↪'Test Subset 1')

# 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')

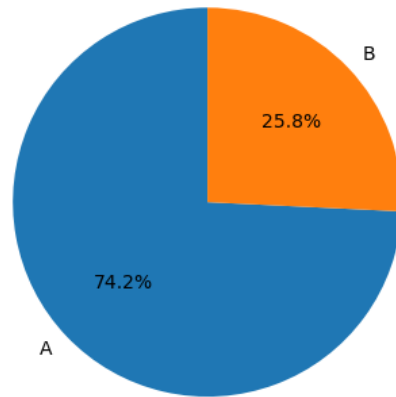
# Terza riga di torte (Subset 3)
```

```
draw_pie(axes[2, 0], train_subset3['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Train Subset 3')  
draw_pie(axes[2, 1], test_subset3['ColonnaAB'].value_counts(normalize=True),  
        ↪ 'Test Subset 3')  
  
# Regolare lo spaziamento tra i subplots  
plt.tight_layout()  
  
# Mostrare il grafico  
plt.show()
```

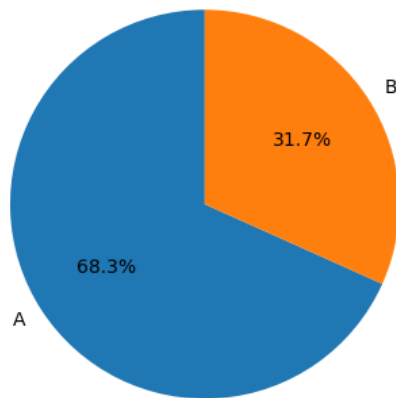
Train Subset 1



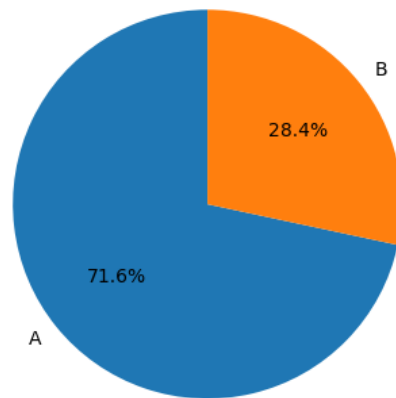
Test Subset 1



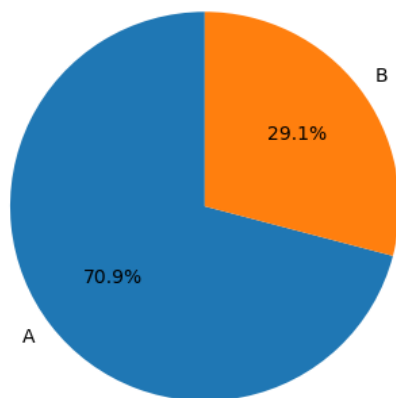
Train Subset 2



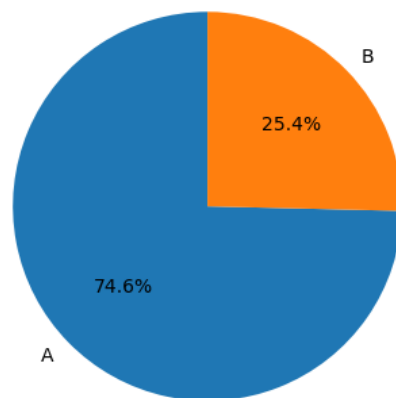
Test Subset 2



Train Subset 3



Test Subset 3



Questo codice esegue le seguenti operazioni:

Divide ciascun subset in training set e test set: Ogni subset viene suddiviso in un training set (80% dei dati) e un test set (20% dei dati). Questo è fatto per avere un set di dati separato (test set) per valutare le prestazioni del modello che sarà addestrato sul training set.

Crea un grafico con 6 torte: Per ogni training set e test set, crea un grafico a torta che mostra la percentuale di 'A' e 'B'.

Mostra il grafico: Infine, mostra i grafici a torta.

In sintesi, questo codice è un esempio di come si potrebbe eseguire un'analisi statistica su diversi training set e test set estratti da diversi subset di un DataFrame e visualizzare i risultati con dei grafici a torta. Questo può essere utile per confrontare la distribuzione di categorie in diversi training set e test set di un dataset.

## 1.3 OUTLIERS

### 1.3.1 Identificazione degli Outliers in un DataFrame: Un Esempio con la Regola delle 3 Sigma

```
[7]: 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()

# 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
```

```
[7]:      Valori
11  100000000
```

Questo codice esegue le seguenti operazioni:

Crea un DataFrame di esempio: Crea un DataFrame di pandas con una colonna chiamata 'Valori' che contiene una lista di numeri, alcuni dei quali sono molto più grandi o più piccoli degli altri (outliers).

Calcola la media e la deviazione standard: Calcola la media e la deviazione standard dei valori nel DataFrame.

Identifica gli outliers: Identifica gli outliers come quei valori che sono a più di 3 deviazioni standard (sigma) dalla media. Questo è un metodo comune per identificare gli outliers in un set di dati.

In sintesi, questo codice è un esempio di come si potrebbe identificare gli outliers in un set di dati utilizzando la regola delle 3 sigma. Gli outliers identificati vengono poi salvati in un nuovo DataFrame chiamato 'outliers'.

### 1.3.2 Visualizzazione di Outliers in un Grafico a Dispersione: Un Esempio con Media e Deviazione Standard

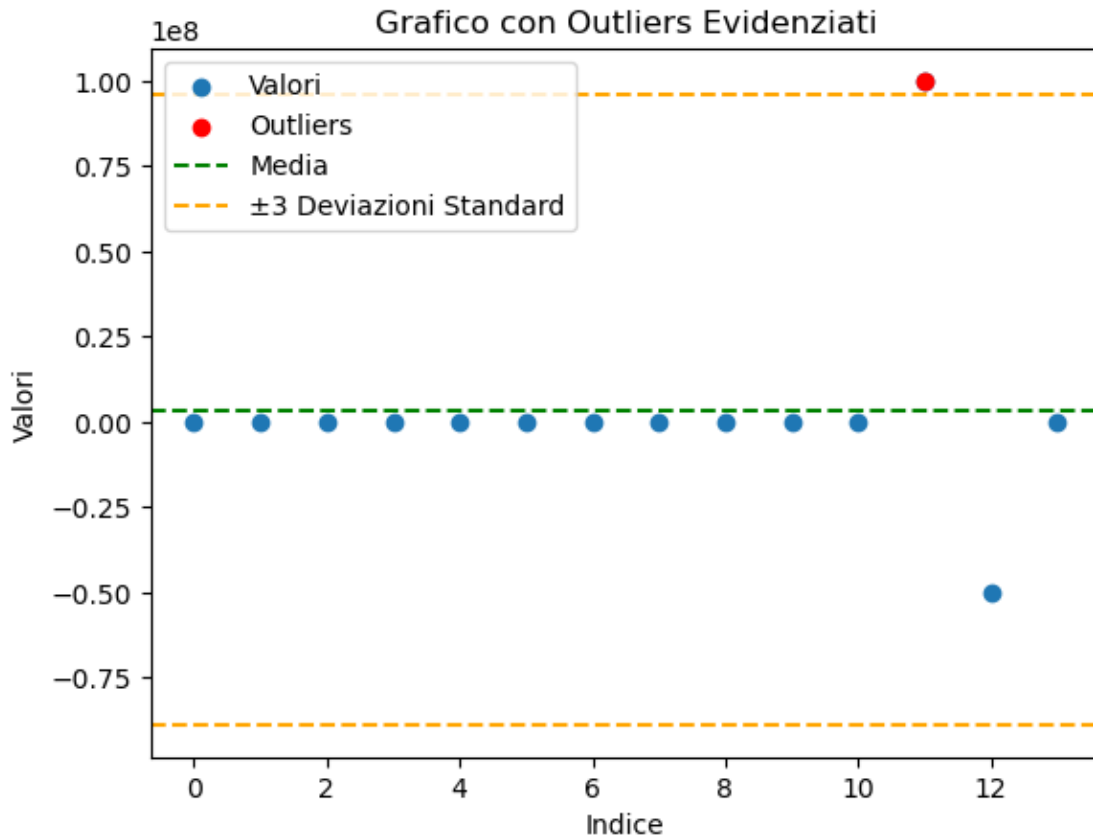
```
[8]: # Crea un grafico a dispersione
plt.scatter(df.index, df['Valori'], label='Valori')

# 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 Deviazioni Standard')
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()
```



Questo codice esegue le seguenti operazioni:

Crea un grafico a dispersione: Crea un grafico a dispersione con l'indice del DataFrame sull'asse x e i valori sull'asse y.

Evidenzia gli outliers nel grafico: Gli outliers vengono evidenziati nel grafico a dispersione con un colore diverso.

Aggiunge la media e la deviazione standard al grafico: Aggiunge linee orizzontali al grafico per indicare la media e la deviazione standard ( $\pm 3$  sigma).

Aggiunge etichette e legenda al grafico: Aggiunge etichette agli assi, un titolo al grafico e una legenda che indica cosa rappresentano i diversi colori e linee.

Mostra il grafico: Infine, mostra il grafico.

In sintesi, questo codice è un esempio di come si potrebbe visualizzare un set di dati e gli outliers associati in un grafico a dispersione, con linee aggiunte per indicare la media e la deviazione standard.

### 1.3.3 Identificazione degli Outliers in un DataFrame Multidimensionale: Un Esempio con la Regola delle k Sigma

```
[6]: 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,
↪1000000000],
        '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 #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] < mean_value - k * std_dev)
    print(df)
```

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1
0	1	2	5	1	False
1	2000	4	10	-20000000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	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
8	2500000	50000	125	20005	False
9	300000000	60	150	30	True
10	1000000000	200	500000	10000	False

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1	\
0	1	2	5	1	False	



1	2000	4	10	-20000000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	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
8	2500000	50000	125	20005	False
9	300000000	60	150	30	True
10	100000000	200	500000	10000	False

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

Outlier_Feature2 Outlier_Feature3		
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	True	False
9	False	False
10	False	True

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1 \
0	1	2	5	1	False
1	2000	4	10	-20000000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	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
8	2500000	50000	125	20005	False
9	300000000	60	150	30	True
10	100000000	200	50000	10000	False

	Outlier_Feature2	Outlier_Feature3	Outlier_Feature4
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	True
4	False	False	False
5	False	False	False
6	False	False	False
7	False	False	False
8	True	False	False
9	False	False	False
10	False	True	False

Questo codice esegue le seguenti operazioni:

Crea un DataFrame di esempio: Crea un DataFrame di pandas con quattro colonne (o “features”) che contengono una lista di numeri, alcuni dei quali sono molto più grandi o più piccoli degli altri (outliers).

Definisce il numero minimo di features che devono superare la soglia per considerare un dato un outlier: Questo è un parametro che può essere modificato in base alle esigenze specifiche dell’analisi.

Crea una lista per salvare gli indici degli outliers: Questa lista sarà popolata con gli indici dei dati che sono considerati outliers.

Itera su ogni feature: Per ogni feature nel DataFrame, calcola la media e la deviazione standard, e identifica gli outliers. Un outlier è definito come un valore che è a più di k deviazioni standard (sigma) dalla media. Questo è un metodo comune per identificare gli outliers in un set di dati.

Identifica gli outliers per ciascuna feature: Aggiunge una nuova colonna al DataFrame per ogni feature, indicando se ogni valore è un outlier per quella feature.

Stampa il DataFrame: Infine, stampa il DataFrame dopo ogni iterazione. Questo ti permette di vedere come il DataFrame cambia dopo ogni iterazione.

In sintesi, questo codice è un esempio di come si potrebbe identificare gli outliers in un DataFrame utilizzando la regola delle k sigma. Gli outliers vengono identificati per ogni feature e vengono aggiunte nuove colonne al DataFrame per indicare gli outliers.

### 1.3.4 Identificazione e Filtraggio degli Outliers in un DataFrame Multidimensionale: Un Esempio con la Regola delle k Sigma

```
[5]: # 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)
print(df)
```

	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	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
8	2500000	50000	125	20005	False
9	300000000	60	150	30	False
10	100000000	200	500000	10000	False

Questo codice esegue le seguenti operazioni:

Calcola il numero di features che superano la soglia per ogni riga: Aggiunge una nuova colonna al DataFrame che conta il numero di features che sono considerate outliers per ogni riga.

Filtra i dati per mantenere solo le righe con almeno il numero minimo di features superanti la soglia: Crea un nuovo DataFrame 'outliers' che contiene solo le righe del DataFrame originale in cui il numero di features che superano la soglia è maggiore o uguale al numero minimo di features impostato.

Aggiunge una colonna che indica se il record è un outlier o meno: Aggiunge una nuova colonna al DataFrame originale che indica se ogni riga è considerata un outlier o meno.

Rimuove le colonne ausiliarie: Rimuove le colonne ausiliarie che sono state aggiunte durante il processo di identificazione degli outliers.

Stampa il DataFrame: Infine, stampa il DataFrame.

In sintesi, questo codice è un esempio di come si potrebbe identificare gli outliers in un DataFrame multidimensionale utilizzando la regola delle k sigma e mantenendo solo le

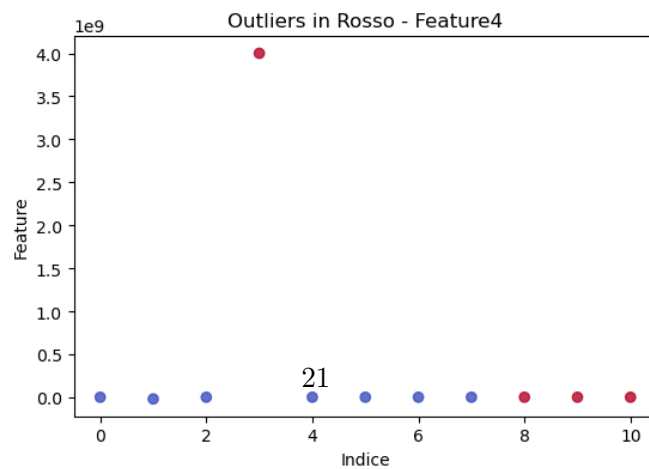
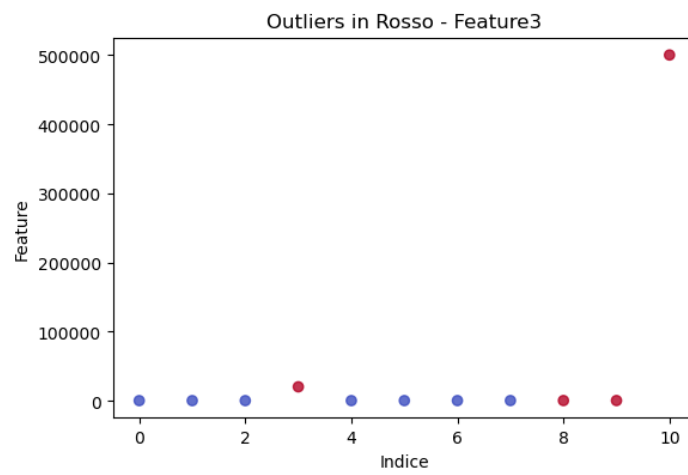
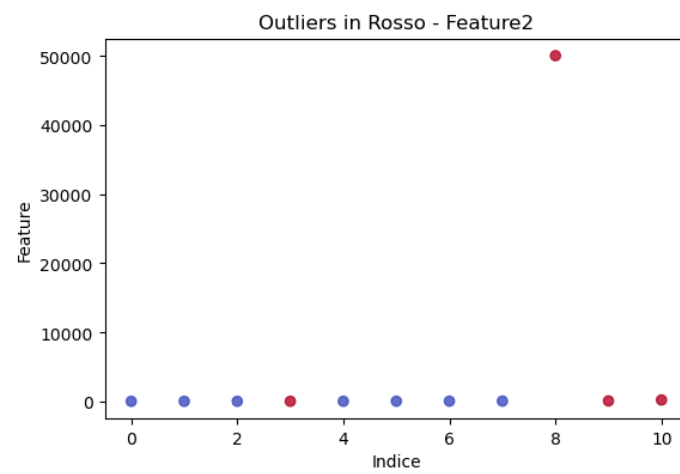
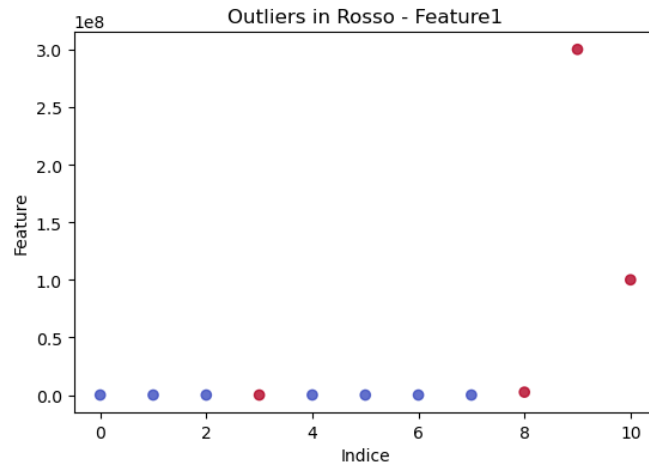
righe che hanno un numero sufficiente di features che superano la soglia.

### 1.3.5 Visualizzazione di Outliers in un DataFrame Multidimensionale: Un Esempio con Grafici a Dispersione

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



Questo codice esegue le seguenti operazioni:

Organizza i grafici in una matrice: Crea una griglia di grafici con una colonna e un numero di righe pari al numero di features nel DataFrame (escludendo la colonna 'Is\_Outlier').

Crea un grafico a dispersione per ogni feature: Per ogni feature nel DataFrame, crea un grafico a dispersione con l'indice del DataFrame sull'asse x e i valori della feature sull'asse y. Gli outliers sono colorati in rosso, mentre i dati normali sono colorati in blu.

Mostra il grafico: Infine, mostra i grafici a dispersione.

In sintesi, questo codice è un esempio di come si potrebbe visualizzare un DataFrame multidimensionale e gli outliers associati in una serie di grafici a dispersione. Questo può essere utile per esaminare visivamente la distribuzione dei dati e la posizione degli outliers in ciascuna feature.

### 1.3.6 Visualizzazione di Outliers in un DataFrame Multidimensionale: Un Esempio con Grafici a Dispersione

```
[12]: # Elimina le righe corrispondenti agli outliers quelli che hanno almeno una
      ↪ features fuoriscala
df_filtered = df[df['Is_Outlier'] == False]
df_filtered
```

```
[12]:
```

	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

Questo codice esegue le seguenti operazioni:

Organizza i grafici in una matrice: Crea una griglia di grafici con una colonna e un numero di righe pari al numero di features nel DataFrame (escludendo la colonna 'Is\_Outlier').

Crea un grafico a dispersione per ogni feature: Per ogni feature nel DataFrame, crea un grafico a dispersione con l'indice del DataFrame sull'asse x e i valori della feature sull'asse y. Gli outliers sono colorati in rosso, mentre i dati normali sono colorati in blu.

Mostra il grafico: Infine, mostra i grafici a dispersione.

In sintesi, questo codice è un esempio di come si potrebbe visualizzare un DataFrame multidimensionale e gli outliers associati in una serie di grafici a dispersione. Questo può essere utile per esaminare visivamente la distribuzione dei dati e la posizione degli outliers in ciascuna feature. Questo codice esegue le seguenti operazioni:

Filtra il DataFrame per rimuovere gli outliers: Crea un nuovo DataFrame `'df_filtered'` che contiene solo le righe del DataFrame originale in cui la colonna `'Is_Outlier'` è False. In altre parole, rimuove tutte le righe che sono state identificate come outliers.

In sintesi, questo codice è un esempio di come si potrebbe filtrare un DataFrame per rimuovere gli outliers. Dopo l'esecuzione di questo codice, `'df_filtered'` conterrà solo i dati che non sono stati identificati come outliers.