Splitting Dataset - Outliers

Componenti del gruppo:

Justin Cadena
Francesco Miraglia
Zhou Zencheng

Indice:

1) Splitting Dataset, 3
2) Outliers, 6

1 Splitting Dataset

1.1 Train test

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Genera dati di esempio per il DataFrame
data = {
    'feature1': [1, 2, 3, 4, 5],
    'feature2': [10, 20, 30, 40, 50],
    'target_column': [0, 1, 0, 1, 1]  # Supponiamo che 'target_column' sia il␣
 ↪target
}

# Creazione del DataFrame
df = pd.DataFrame(data)

# Dividi le features (X) e il target (y)
X = df.drop(columns=['target_column'])
y = df['target_column']

# Esegui lo splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Ora puoi utilizzare X_train, X_test, y_train e y_test per addestrare e␣
 ↪valutare il tuo modello
```

[2]:
```
# Stampa le dimensioni dei dataset di addestramento e test
print("Dimensioni di X_train:", X_train.shape)
print("Dimensioni di X_test:", X_test.shape)
print("Dimensioni di y_train:", y_train.shape)
print("Dimensioni di y_test:", y_test.shape)

# Stampa i primi elementi di ciascun dataset
print("\nPrimi 5 elementi di X_train:")
print(X_train.head())

print("\nPrimi 5 elementi di X_test:")
print(X_test.head())

print("\nPrimi 5 elementi di y_train:")
print(y_train.head())
```

```
print("\nPrimi 5 elementi di y_test:")
print(y_test.head())
```

```
Dimensioni di X_train: (4, 2)
Dimensioni di X_test: (1, 2)
Dimensioni di y_train: (4,)
Dimensioni di y_test: (1,)

Primi 5 elementi di X_train:
   feature1  feature2
4         5        50
2         3        30
0         1        10
3         4        40

Primi 5 elementi di X_test:
   feature1  feature2
1         2        20

Primi 5 elementi di y_train:
4    1
2    0
0    0
3    1
Name: target_column, dtype: int64

Primi 5 elementi di y_test:
1    1
Name: target_column, dtype: int64
```

1.2 Cross-validation

[3]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Creiamo un dataset fittizio
data = {'feature1': [1, 2, 3, 4, 5],
        'feature2': [10, 20, 30, 40, 50],
        'target': [0, 1, 0, 1, 0]}

df = pd.DataFrame(data)

# Creiamo un modello di regressione lineare
model = LinearRegression()

# Eseguiamo la validazione incrociata
scores = cross_val_score(model, df[['feature1', 'feature2']], df['target'],␣
 ↪cv=5, scoring='neg_mean_squared_error')
```

```
print("Mean squared error (MSE) per fold:")
print(-scores)
```

```
Mean squared error (MSE) per fold:
[1.         0.73469388 0.25       0.73469388 1.        ]
```

1.3 Stratified Sampling (Campionamento stratificato)

[4]:
```
import pandas as pd

# Creiamo un dataset fittizio
data = {'feature1': [1, 2, 3, 4, 5],
        'feature2': [10, 20, 30, 40, 50],
        'target': ['A', 'B', 'A', 'B', 'A']}

df = pd.DataFrame(data)

# Eseguiamo il campionamento stratificato
sample = df.groupby('target', group_keys=False).apply(lambda x: x.sample(n=2))

print("Campionamento stratificato:")
print(sample)
```

```
Campionamento stratificato:
   feature1  feature2 target
2         3        30      A
4         5        50      A
3         4        40      B
1         2        20      B
```

1.4 Time Series Split

[5]:
```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit

# Creiamo un dataset fittizio
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4, 5, 6])

# Eseguiamo la divisione per serie temporali
tscv = TimeSeriesSplit()
for i, (train_index, test_index) in enumerate(tscv.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test: index={test_index}")
```

```
Fold 0:
```

```
  Train: index=[0]
  Test: index=[1]
Fold 1:
  Train: index=[0 1]
  Test: index=[2]
Fold 2:
  Train: index=[0 1 2]
  Test: index=[3]
Fold 3:
  Train: index=[0 1 2 3]
  Test: index=[4]
Fold 4:
  Train: index=[0 1 2 3 4]
  Test: index=[5]
```

---

2 Outliers

2.1 Rilevazione degli Outliers in un DataFrame

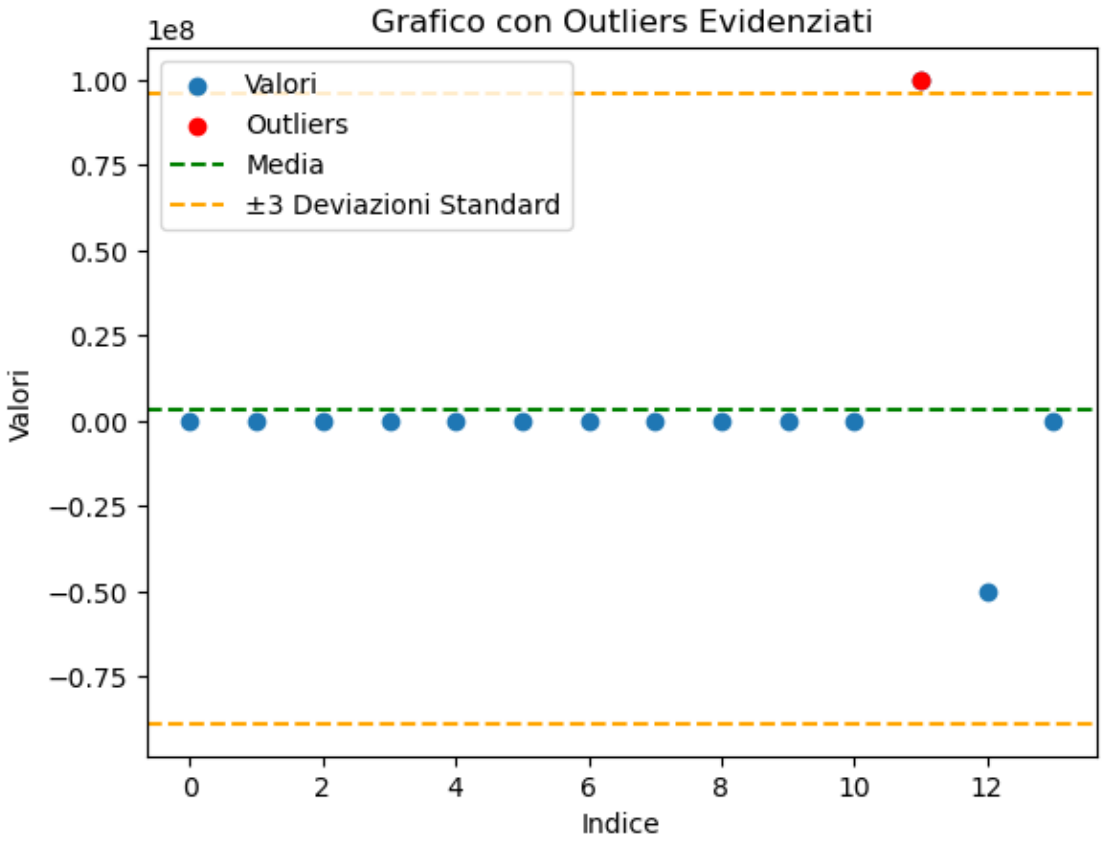
[1]:
```
import pandas as pd
import matplotlib.pyplot as plt

# 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
```

[1]:
```
       Valori
11  100000000
```

2.2 Grafico a dispersione

[2]:
```
# 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()
```



2.3 Z-score

2.4 Metodo Z-score

Il metodo Z-score è una misura statistica che indica di quanti deviazioni standard un determinato dato si discosta dalla media di una distribuzione (o valore medio). Questo metodo è utile per standardizzare e confrontare punti dati provenienti da diverse distribuzioni.

[3]:
```
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 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)
df
```

[3]:

	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  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
```

2.5 Rilevazione degli Outliers in un DataFrame

[4]:
```
# 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
```

[4]:
```
      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 Features che superano la soglia per ogni riga

[5]:
```
#Calcola il numero di features che superano la soglia per ogni riga
df['Num_Outliers'] = df.filter(like='Outlier_.').sum(axis=1)
df
```

[5]:

	Feature1	Feature2	Feature3	Feature4	Is_Outlier	Num_Outliers
0	1	2	5	1	False	0.0
1	2000	4	10	-20000000	False	0.0
2	3	6	15	3	False	0.0
3	4	8	20000	4000000000	True	0.0
4	50000	10	25	5	False	0.0
5	10	20	50	10	False	0.0
6	15	30	75	15	False	0.0
7	20	40	100	20	False	0.0
8	2500000	50000	125	20005	True	0.0
9	300000000	60	150	30	True	0.0
10	100000000	200	500000	10000	True	0.0

2.7 Dati che mantengono le features che superano la soglia

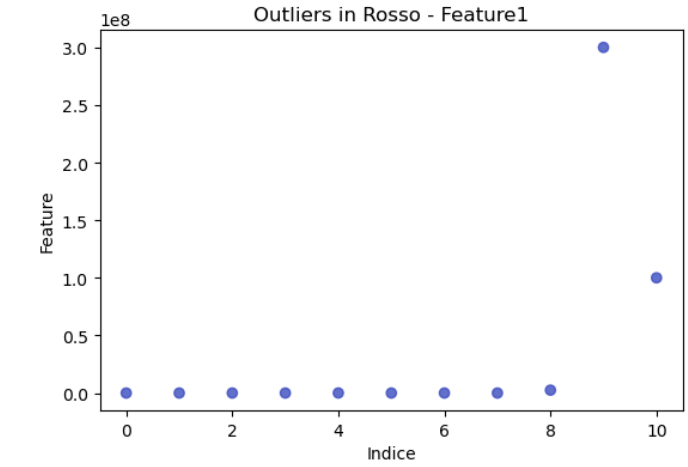
[6]:
```
# Filtra i dati per mantenere solo le righe con almeno il numero minimo di␣
 ↪features superanti la soglia
outliers = df[df['Num_Outliers'] >= min_features_threshold]

# 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
```

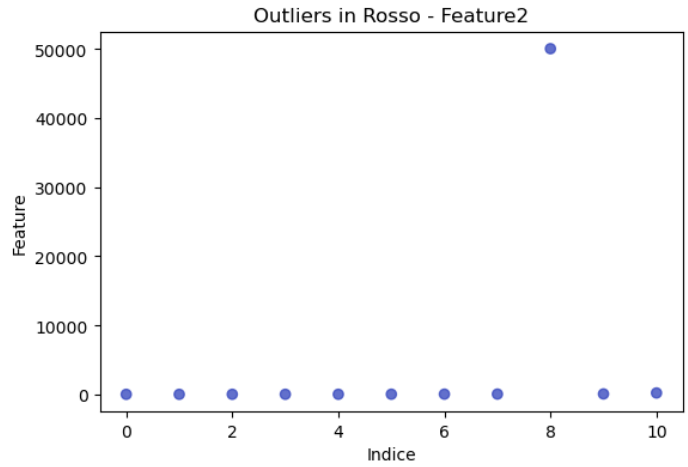
[6]:

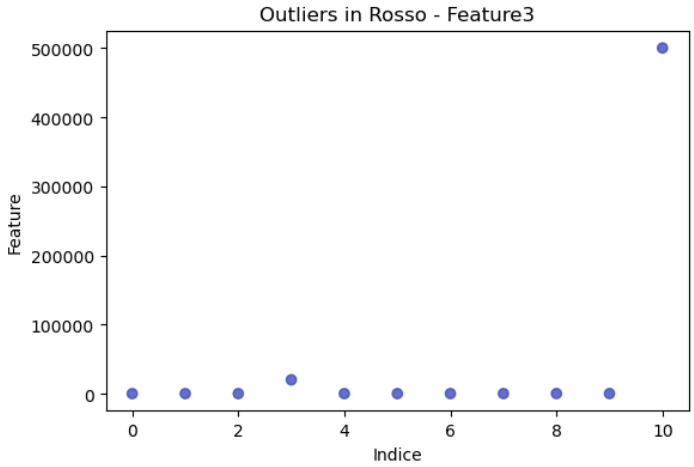
	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	2000	4	10	-20000000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	False
4	50000	10	25	5	False
5	10	20	50	10	False
6	15	30	75	15	False
7	20	40	100	20	False
8	2500000	50000	125	20005	False
9	300000000	60	150	30	False
10	100000000	200	500000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

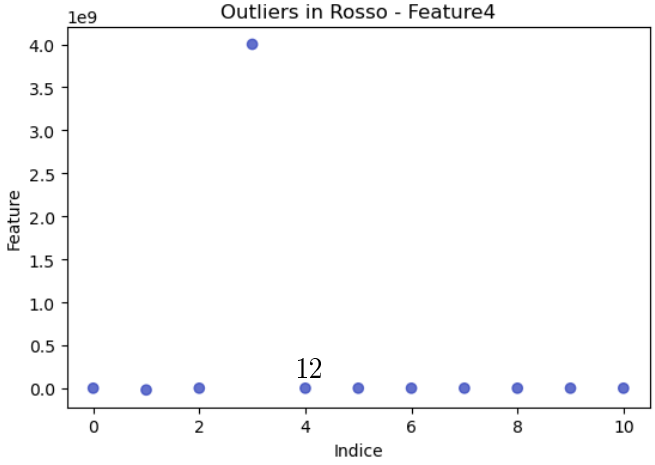
2.8 Grafico a matrice

[7]:
```
# 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()
```


Outliers in Rosso - Feature1
1e8
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Feature
0
2
4
6
8
10
Indice


Outliers in Rosso - Feature2
50000
40000
30000
20000
10000
0
Feature
0
2
4
6
8
10
Indice


Outliers in Rosso - Feature3
500000
400000
300000
200000
100000
0
Feature
0
2
4
6
8
10
Indice


Outliers in Rosso - Feature4
1e9
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Feature
0
2
4
6
8
10
Indice

2.9 Eliminazione di righe che hanno una riga fuori scala

[8]:
```
#Elimina le righe corrispondenti agli outliers quelli che hanno almeno una␣
 ↪feature fuori scala
df_filtered = df[df['Is_Outlier'] == False]
df_filtered
```

[8]:

	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

---

3 Scarto interquartile (IQR)

È un indice di dispersione utilizzato in statistica per misurare quanto i valori di un insieme di dati si discostano dalla loro mediana.

3.1 Calcolo dell'IQR per un array

[9]:
```
import numpy as np

# Definizione di un array di dati
data = np.array([14, 19, 20, 22, 24, 26, 27, 30, 30, 31, 36, 38, 44, 47])

# Calcolo del terzo quartile (Q3) e del primo quartile (Q1) utilizzando la␣
 ↪funzione np.percentile()
q3, q1 = np.percentile(data, [75, 25])

# Calcolo dell'Interquartile Range (IQR) come differenza tra Q3 e Q1
iqr = q3 - q1

# Stampare l'Interquartile Range
print("Interquartile Range:", iqr)
```

```
Interquartile Range: 12.25
```

3.2 Calcolo dell'IQR per una colonna di un DataFrame:

[10]:
```
import pandas as pd

# Creazione di un DataFrame con quattro colonne: 'rating', 'points', 'assists',␣
↪'rebounds'
data1 = pd.DataFrame({'rating': [90, 85, 82, 88, 94, 90, 76, 75, 87, 86],
                      'points': [25, 20, 14, 16, 27, 20, 12, 15, 14, 19],
                      'assists': [5, 7, 7, 8, 5, 7, 6, 9, 9, 5],
                      'rebounds': [11, 8, 10, 6, 6, 9, 6, 10, 10, 7]})

# Calcolo del terzo quartile (q75) e del primo quartile (q25) della colonna␣
↪'points' utilizzando la funzione np.percentile()
q75, q25 = np.percentile(data1['points'], [75, 25])

# Calcolo dell'Interquartile Range (IQR) come differenza tra q75 e q25
iqr = q75 - q25

# Stampa dell'Interquartile Range per la colonna 'points'
print("Interquartile Range (points column):", iqr)
```

```
Interquartile Range (points column): 5.75
```

3.3 Calcolo dell'IQR per più colonne di un DataFrame

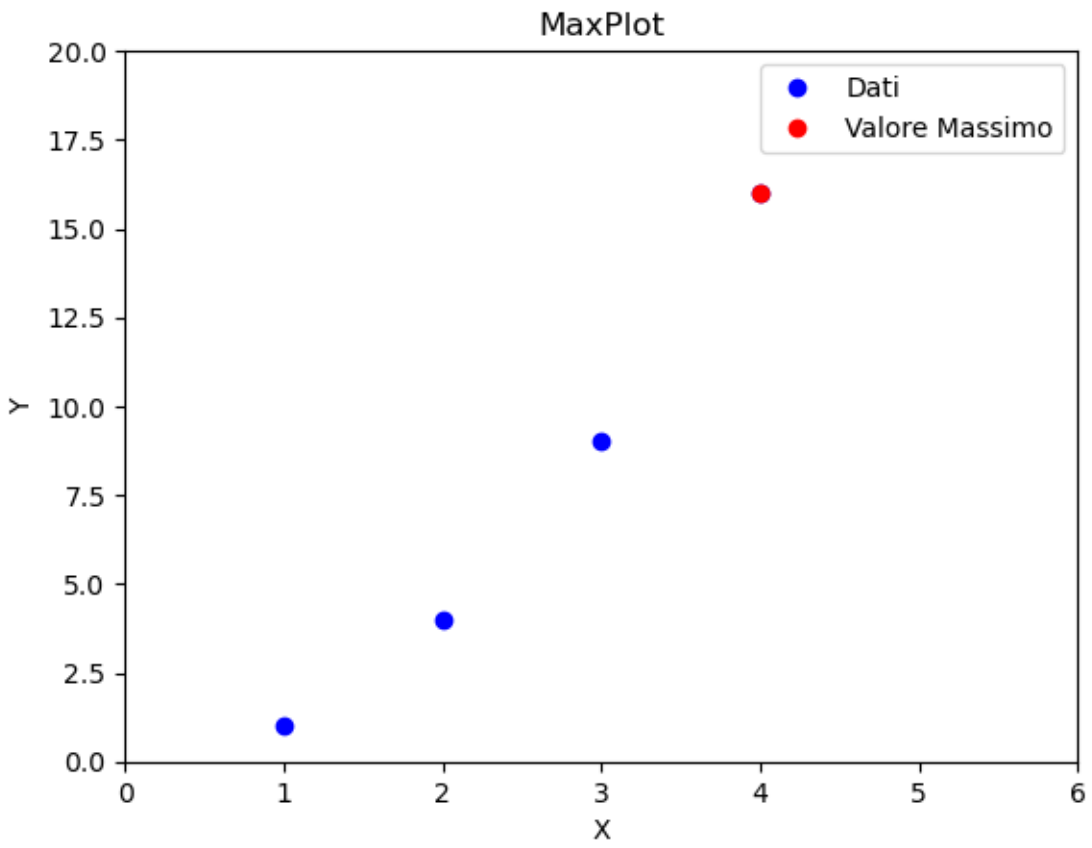
[11]:
```
# Definizione della funzione find_iqr(x) che calcola l'IQR di una serie di dati x
def find_iqr(x):
    return np.subtract(*np.percentile(x, [75, 25]))

# Applicazione della funzione find_iqr alla selezione delle colonne 'rating' e␣
↪'points' del DataFrame df
iqr_values = data1[['rating', 'points']].apply(find_iqr)

# Stampare i valori dell'Interquartile Range per le colonne 'rating' e 'points'
print(iqr_values)
```

```
rating    6.75
points    5.75
dtype: float64
```

4 Maxplot

[12]:
```
import matplotlib.pyplot as plt

# Dati di esempio (x e y)
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
```

```
# Trova l'indice del valore massimo di y
n_max = y.index(max(y))

# Crea un grafico con punti rossi per il valore massimo
plt.plot(x, y, 'bo', label='Dati')
plt.plot(x[n_max], y[n_max], 'ro', label='Valore Massimo')

# Imposta i limiti degli assi
plt.axis((0, 6, 0, 20))

# Etichette degli assi
plt.xlabel('X')
plt.ylabel('Y')

# Titolo del grafico
plt.title('MaxPlot')

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



5 Deviazione standard

[13]:
```python
def calcola_deviazione_standard(lista):
    n = len(lista)

    # Calcola la media
    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, 5]
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: 1.4142135623730951
```