Splitting Dataset & Outliers

Componenti del gruppo:

Justin Cadena
Francesco Miraglia
Zhou Zencheng

Indice:

1 Splitting Dataset

---

2 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
```

2.1 Generazione e Suddivisione dei Dati

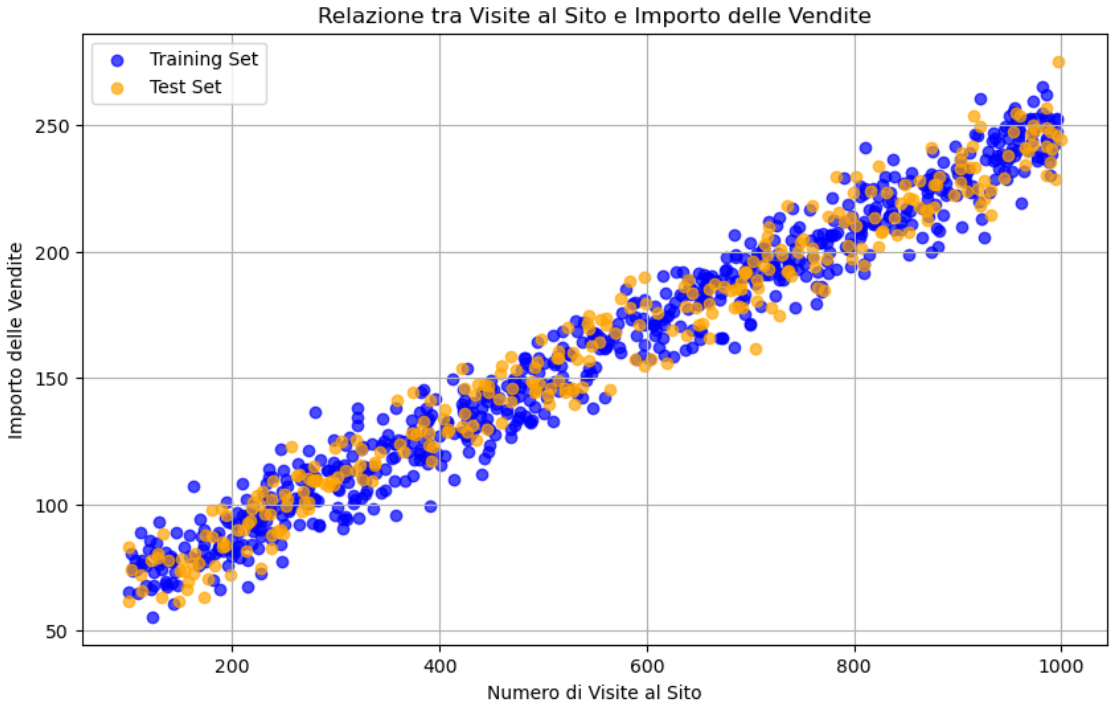
[3]:
```
import numpy as np
from sklearn.model_selection import train_test_split

# 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,)
```

2.2 Analisi della Relazione tra Visite di un Sito e Importo delle Vendite

[4]:
```
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_state=42)

# 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)
```

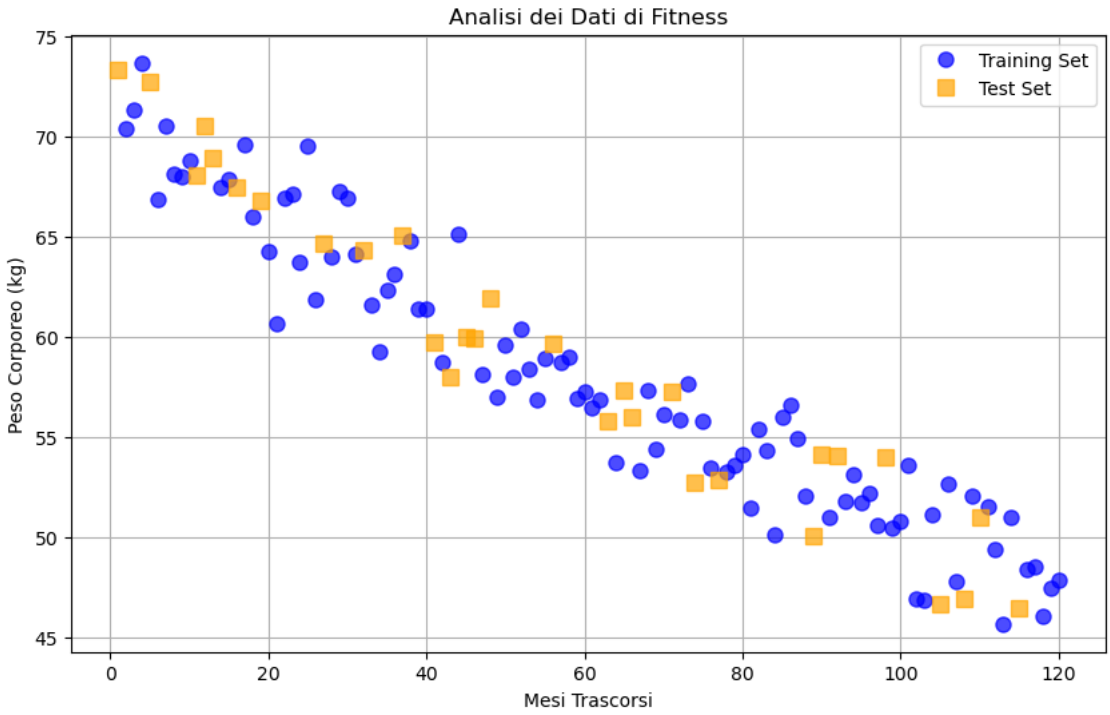



```
Dimensioni del Training Set (visite al sito e importo delle vendite): (700,)
(700,)
Dimensioni del Test Set (visite al sito e importo delle vendite): (300,) (300,)
```

2.3 Analisi dei Dati di Fitness: Relazione tra Mesi Trascorsi e Peso Corporeo

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Creazione di dati casuali per mesi trascorsi e peso corporeo
np.random.seed(0)
n=120
mesi_trascorsi = np.arange(1, n+1)
peso_corporeo = 70 - 0.2 * mesi_trascorsi + np.random.normal(0, 2, n)

# Suddivisione del dataset in training set (75%) e test set (25%)
X_train, X_test, y_train, y_test = train_test_split(mesi_trascorsi,␣
↪peso_corporeo, test_size=0.25, random_state=42)

# Creazione di un grafico a linee
plt.figure(figsize=(10, 6))
plt.plot(X_train, y_train, label='Training Set', marker='o', color='blue',␣
↪linestyle='', markersize=8,alpha=0.7)
```

```
plt.plot(X_test, y_test, label='Test Set', marker='s', color='orange',␣
 ↪linestyle='', markersize=8,alpha=0.7)
plt.xlabel('Mesi Trascorsi')
plt.ylabel('Peso Corporeo (kg)')
plt.title('Analisi dei Dati di Fitness')
plt.legend()
plt.grid(True)
plt.show()

# Stampare le dimensioni dei training set e test set
print("Dimensioni del Training Set (mesi trascorsi e peso corporeo):", X_train.
 ↪shape, y_train.shape)
print("Dimensioni del Test Set (mesi trascorsi e peso corporeo):", X_test.shape,␣
 ↪y_test.shape)
```

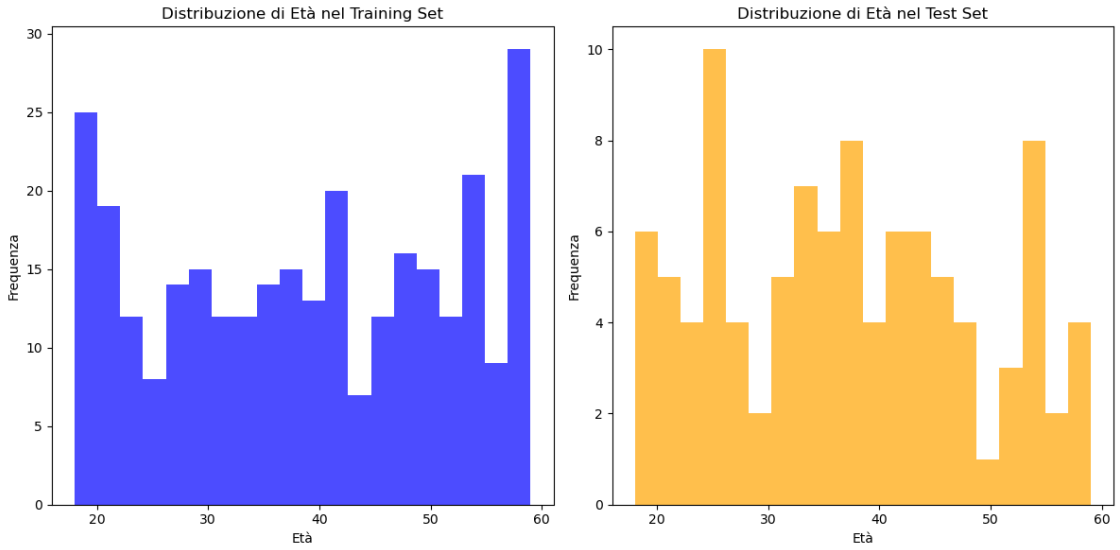



```
Dimensioni del Training Set (mesi trascorsi e peso corporeo): (90,) (90,)
Dimensioni del Test Set (mesi trascorsi e peso corporeo): (30,) (30,)
```

2.4 Confronto delle Distribuzioni di Età

[6]:
```
import numpy as np
import matplotlib.pyplot as plt

# Creazione di dati casuali per età
```

```
np.random.seed(0)
eta_training_set = np.random.randint(18, 60, 300)
eta_test_set = np.random.randint(18, 60, 100)

# Confronto delle distribuzioni di età
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(eta_training_set, bins=20, color='blue', alpha=0.7)
plt.title('Distribuzione di Età nel Training Set')
plt.xlabel('Età')
plt.ylabel('Frequenza')

plt.subplot(1, 2, 2)
plt.hist(eta_test_set, bins=20, color='orange', alpha=0.7)
plt.title('Distribuzione di Età nel Test Set')
plt.xlabel('Età')
plt.ylabel('Frequenza')

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



2.5 Split Stratificato dei Dati per il Training set e il Test set

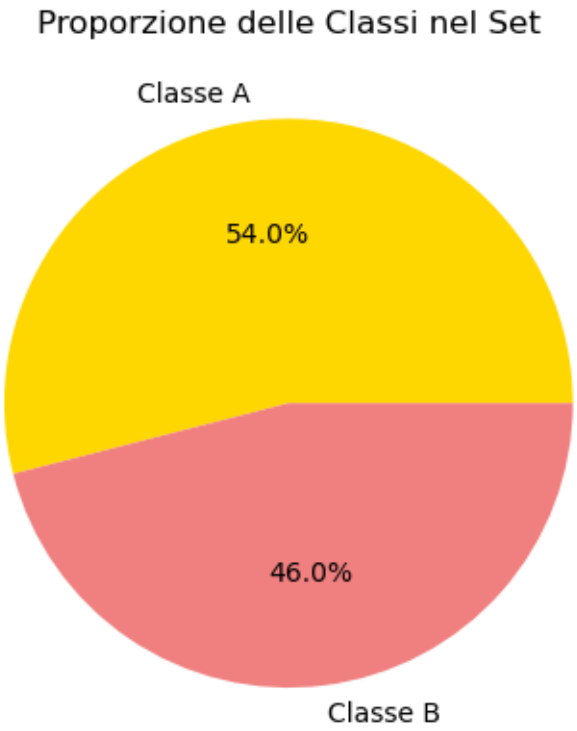
[7]:
```
from sklearn.model_selection import train_test_split
import numpy as np

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
```

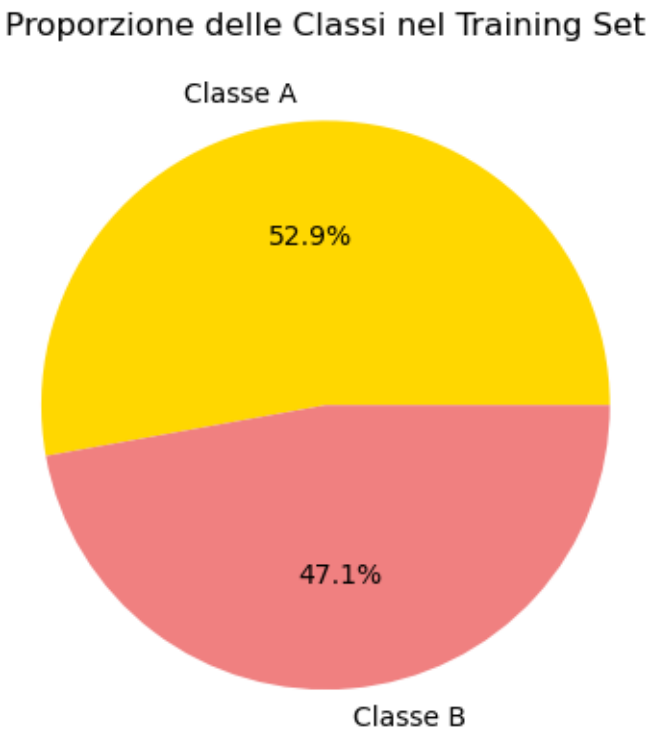
2.6 Grafico di distribuzione delle Classi nel Set

[8]:
```
# Etichette delle classi
labels = ['Classe A', 'Classe B']
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']
# Crea un grafico a torta con etichette
plt.pie([proporzione_classe_A, proporzione_classe_B], labels=labels,␣
 ↪colors=colors, autopct='%1.1f%%')
plt.title('Proporzione delle Classi nel Set')
plt.show()
```



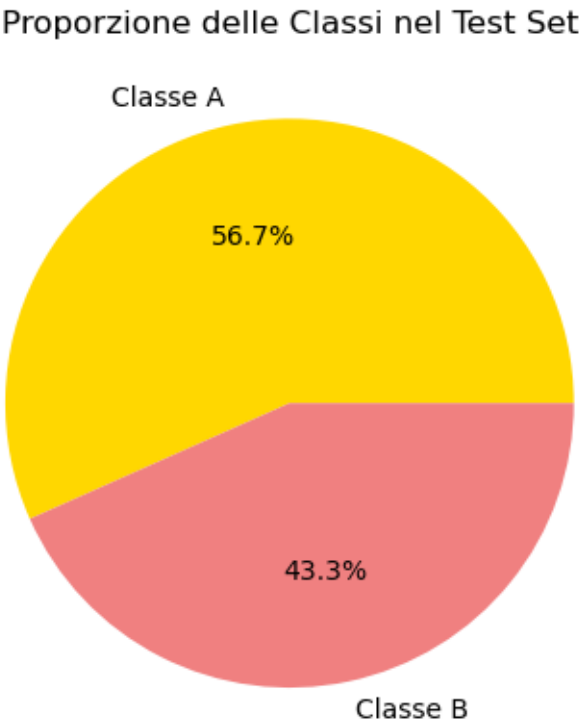
2.7 Grafico di distribuzione delle Classi nel Training Set

[9]:
```
# Etichette delle classi
labels = ['Classe A', 'Classe B']
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']

# Crea un grafico a torta con etichette
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()
```



2.8 Grafico di distribuzione delle Classi nel Test Set

[10]:
```
# Crea un grafico a torta con etichette
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()
```



2.9 Analisi Statistica su Campione Casuale e Dataset

[11]:
```
import random
import numpy as np

dataset=[]
# Creazione di un dataset di 1000 elementi (ad esempio, dati casuali)
popolazione =24000000
for i in range(popolazione):
    dataset.append(random.randint(0, 100000))

campione = int(round(0.3 * popolazione))# Estrazione di un campione casuale␣
 ↪semplice dal dataset
campione_casuale = random.sample(dataset, campione)

# 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:  50000.12
Deviazione standard del campione casuale:  28871.00
Media del dataset completo:  49999.32
Deviazione standard del dataset completo:  28868.78
```

2.10 Creazione di DataFrame con Distribuzione Controllata

[12]:
```
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 = 100000
# 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})
df
```

[12]:

	ColonnaAB
0	A
1	B
2	B
3	A
4	A
...	...
99995	B
99996	B
99997	A
99998	A
99999	A

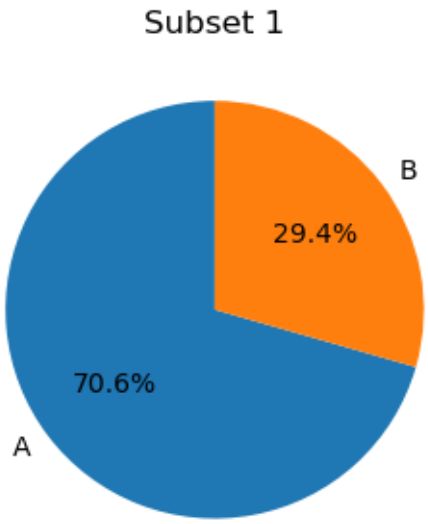
```
[100000 rows x 1 columns]
```

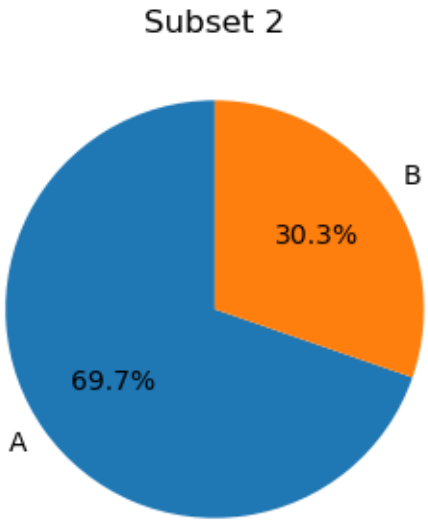
2.11 Creazione di Subset di Dimensioni Simili da un DataFrame

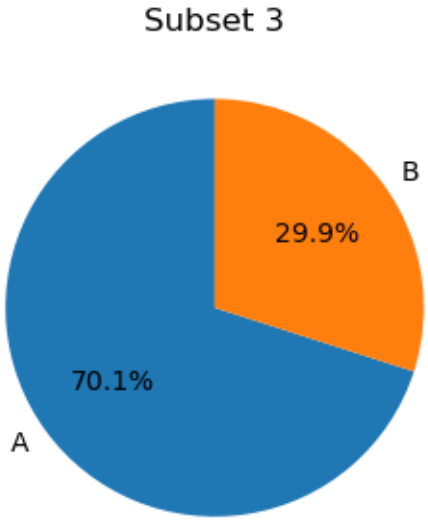
[13]:
```
# Creare tre subset di dimensioni simili
subset1 = df.sample(frac=1/3)
df = df.drop(subset1.index)

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

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

2.12 Analisi delle Distribuzioni delle Classi nei Subset

[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%%', 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()
```

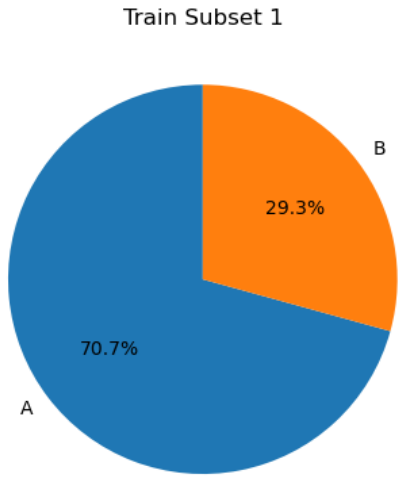

Subset 1
B
29.4%
70.6%
A

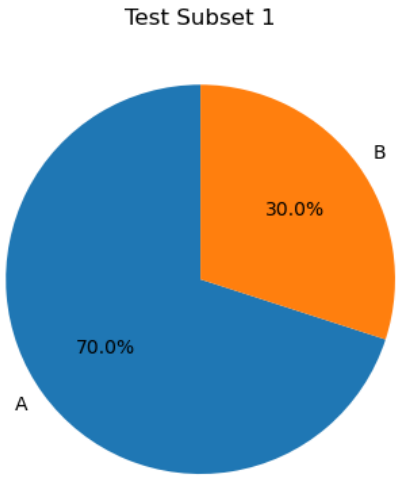

Subset 2
B
30.3%
69.7%
A

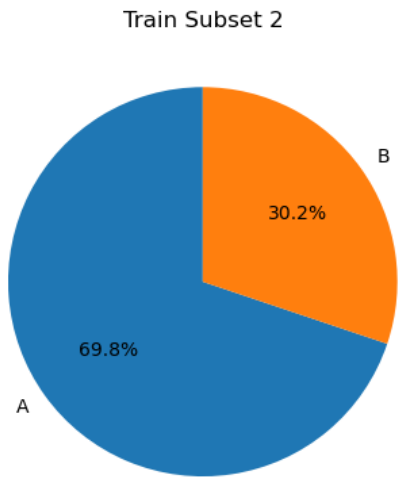

Subset 3
B
29.9%
70.1%
A

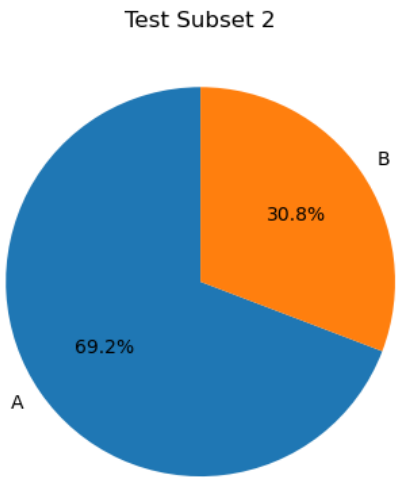
2.13 Divisione dei Subset in Training Set e Test Set con Analisi delle Distribuzioni delle Classi

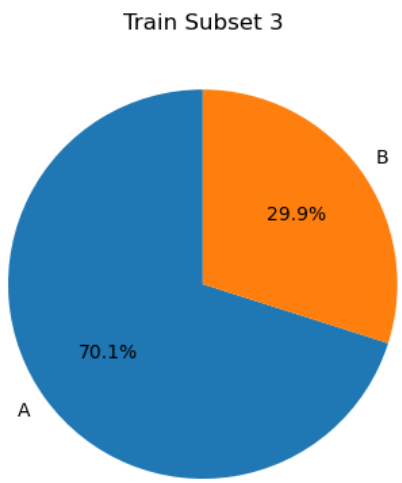
[15]:
```
# Dividere ciascun subset in training set e test set
train_subset1, test_subset1 = train_test_split(subset1, test_size=0.2,␣
↪random_state=42)
train_subset2, test_subset2 = train_test_split(subset2, test_size=0.2,␣
↪random_state=42)
train_subset3, test_subset3 = train_test_split(subset3, test_size=0.2,␣
↪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 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
plt.tight_layout()

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

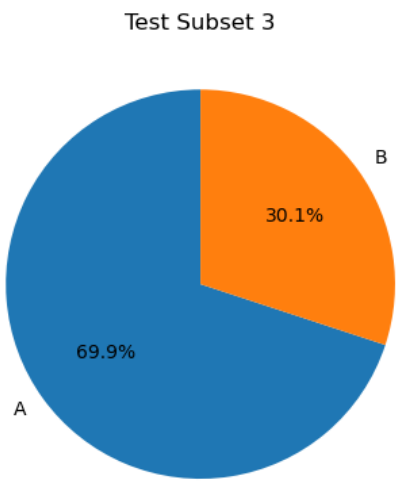

Train Subset 1
B
29.3%
70.7%
A


Test Subset 1
B
30.0%
70.0%
A


Train Subset 2
B
30.2%
69.8%
A


Test Subset 2
B
30.8%
69.2%
A

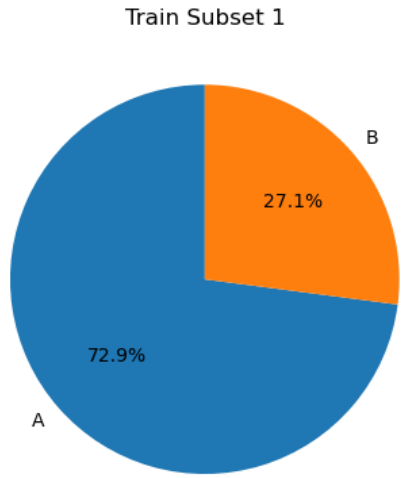

Train Subset 3
B
29.9%
70.1%
A

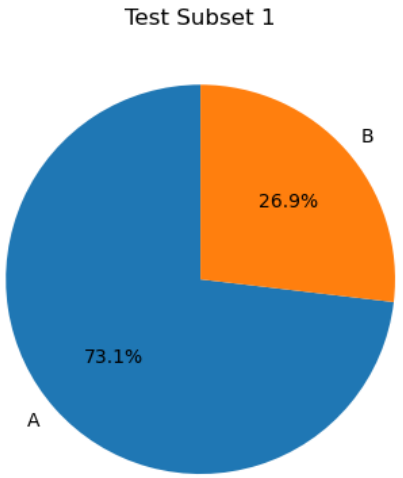

Test Subset 3
B
30.1%
69.9%
A

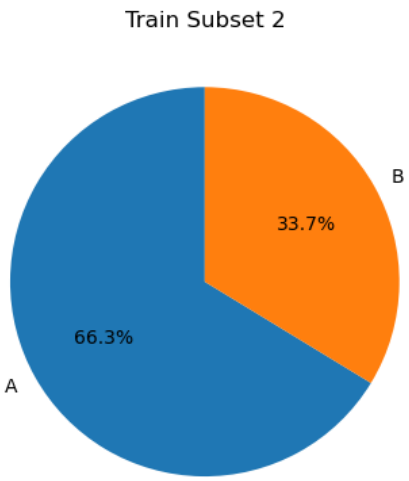
2.14 Divisione dei Subset Stratificati in Training Set e Test Set con Analisi delle Distribuzioni delle Classi

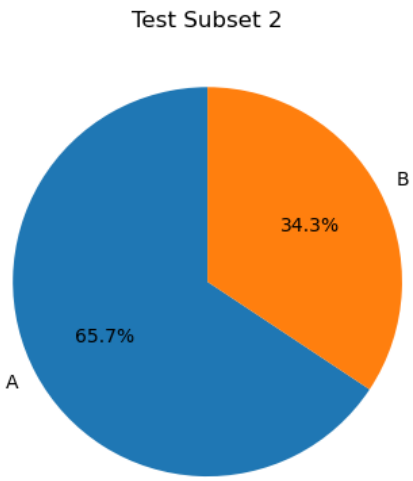
[16]:
```
np.random.seed(41)

# Creare il DataFrame originale
num_elementi = 1000
percentuale_A = 0.7
colonna = np.random.choice(['A', 'B'], size=num_elementi, p=[percentuale_A, 1 -␣
↪percentuale_A])
df = pd.DataFrame({'ColonnaAB': colonna})

# Creare tre subset di dimensioni simili
subset1 = df.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

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

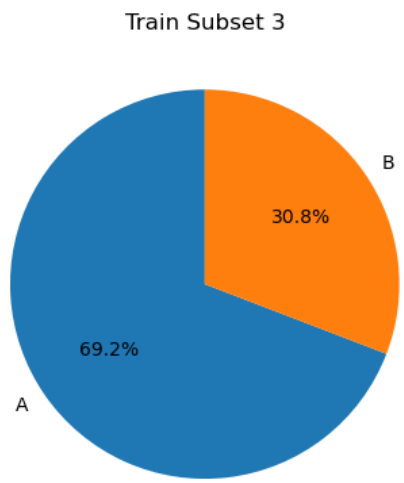
```
draw_pie(axs[1, 1], test_subset2['ColonnaAB'].value_counts(normalize=True),␣
 ↪'Test Subset 2')

# Terza riga di torte (Subset 3)
draw_pie(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
plt.tight_layout()

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

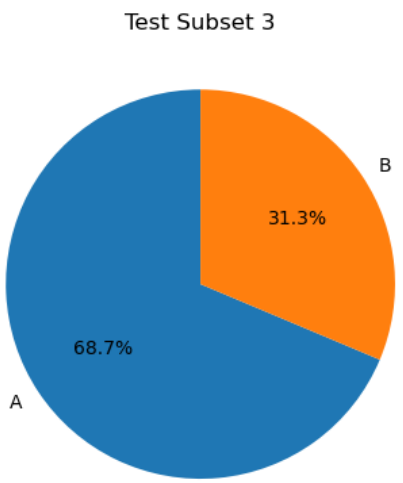

Train Subset 1
B
27.1%
72.9%
A


Test Subset 1
B
26.9%
73.1%
A


Train Subset 2
B
33.7%
66.3%
A


Test Subset 2
B
34.3%
65.7%
A

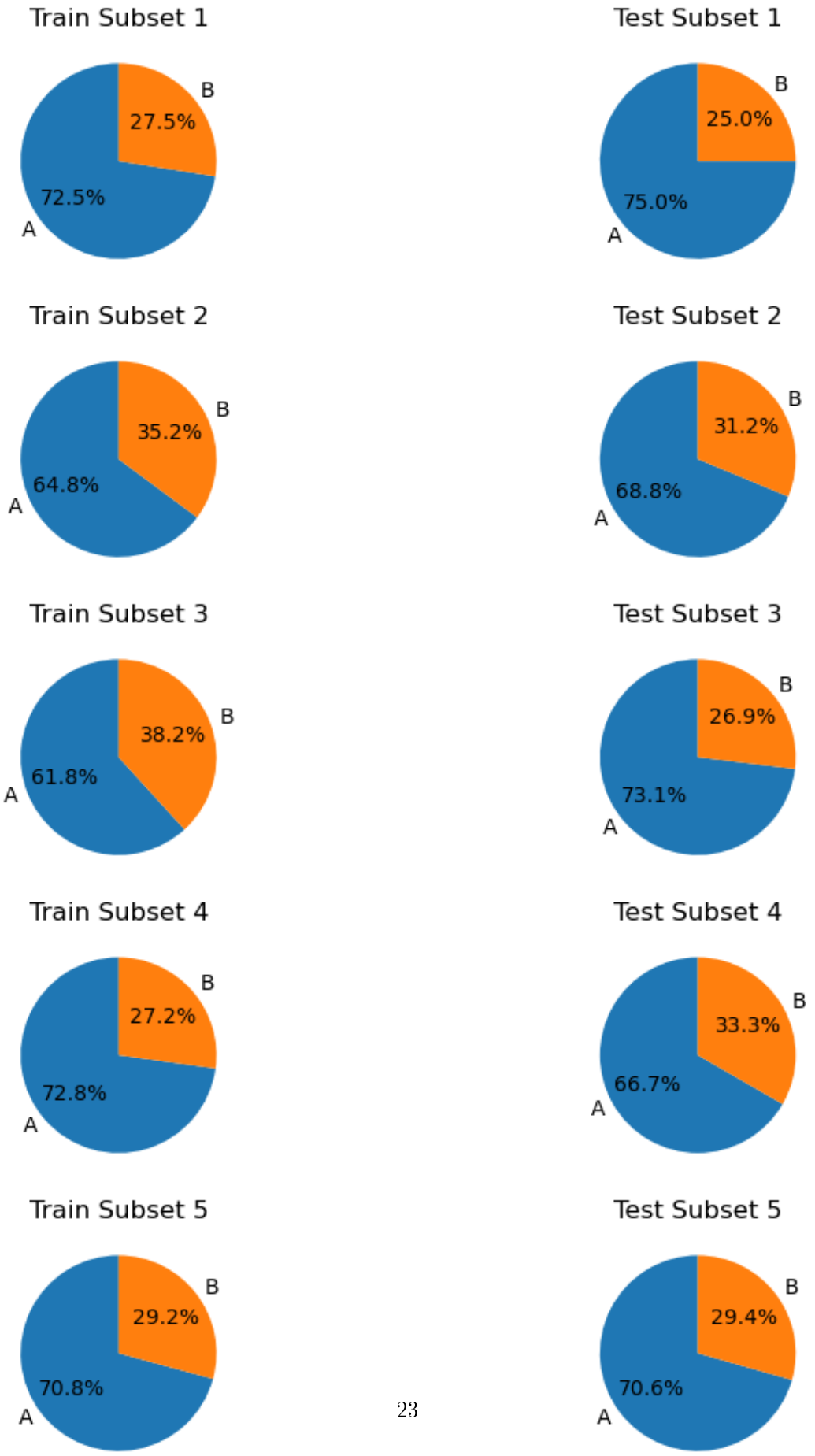

Train Subset 3
B
30.8%
69.2%
A


Test Subset 3
B
31.3%
68.7%
A

2.15 Analisi delle Distribuzioni delle Classi nei Subset con Divisione in Training Set e Test Set

[17]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

# Impostare il seed per la riproducibilità
np.random.seed(41)

# Creare il DataFrame originale
num_elementi = 1000
percentuale_A = 0.7
colonna = np.random.choice(['A', 'B'], size=num_elementi, p=[percentuale_A, 1 -␣
 ↪percentuale_A])
df = pd.DataFrame({'ColonnaAB': colonna})

# Numero di subset desiderato
num_subset = 5

# Creare i subset di dimensioni simili
subset_list = []
for i in range(num_subset):
    subset = df.sample(frac=1/num_subset)
    df = df.drop(subset.index)
    subset_list.append(subset)

# Creare il grafico con 2 torte per ognuno dei N subset
fig, axs = plt.subplots(num_subset, 2, figsize=(10, 2*num_subset))

# Iterare attraverso i subset e disegnare le torte
for i, subset in enumerate(subset_list):
    # Dividere ciascun subset in training set e test set
    train_set, test_set = train_test_split(subset, test_size=0.2,␣
 ↪random_state=42) # posso aggiungere stratify=subset['ColonnaAB']

    # Prima colonna: Training Set
    draw_pie(axs[i, 0], train_set['ColonnaAB'].value_counts(normalize=True),␣
 ↪f'Train Subset {i + 1}')

    # Seconda colonna: Test Set
    draw_pie(axs[i, 1], test_set['ColonnaAB'].value_counts(normalize=True),␣
 ↪f'Test Subset {i + 1}')

# Regolare lo spaziamento tra i subplots
```

```
plt.tight_layout()

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


Train Subset 1
B
27.5%
72.5%
A
Test Subset 1
B
25.0%
75.0%
A
Train Subset 2
B
35.2%
64.8%
A
Test Subset 2
B
31.2%
68.8%
A
Train Subset 3
B
38.2%
61.8%
A
Test Subset 3
B
26.9%
73.1%
A
Train Subset 4
B
27.2%
72.8%
A
Test Subset 4
B
33.3%
66.7%
A
Train Subset 5
B
29.2%
70.8%
A
Test Subset 5
B
29.4%
70.6%
A

3 Extra

---

4 Cross-validation

[18]:
```
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.        ]
```

5 Stratified Sampling (Campionamento stratificato)

[19]:
```
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
0         1        10      A
1         2        20      B
3         4        40      B
```

6 Time Series Split

[20]:
```
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]
```

7 Outliers

7.1 Rilevazione degli Outliers in un Dataframe

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

# 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
```

[1]:
```
30786384.39895254
```

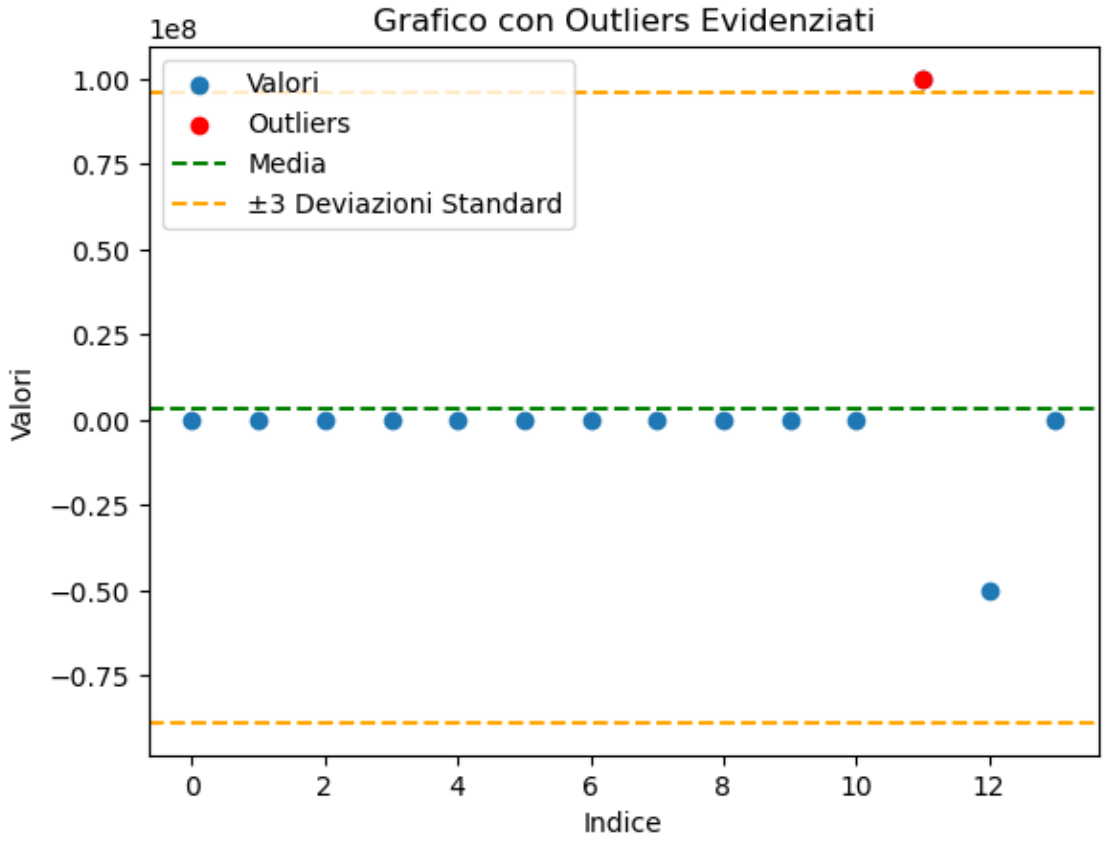
[2]:
```
# Identifica gli outliers considerando ±3 sigma dalla media
outliers = df[(df['Valori'] > mean_value + 3 * std_dev) | (df['Valori'] <␣
 ↪mean_value - 3 * std_dev)]
outliers
```

[2]:
```
       Valori
11  100000000
```

7.2 Grafico a Dispersione

[3]:
```
# 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()
```



8 Metodi per rilevare gli Outliers

9 Z-score

[4]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Crea un DataFrame di esempio con 4 features
data = {'Feature1': [1, 200, 3, 4, 50000, 10, 15, 20, 2500000, 300000000,␣
 ↪100000000],
        'Feature2': [2, 4, 6, 8, 10, 20, 30, 40, 500, 60, 200],
        'Feature3': [5, 10, 15, 20000, 25, 50, 75, 100, 125, 150, 500000],
        'Feature4': [1, -200000, 3, 4000000000, 5, 10, 15, 20, 200, 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=3 #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
```

[4]:

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1	Outlier_Feature2	Outlier_Feature3	Outlier_Feature4
0	1	2	5	1	False	False	False	False
1	200	4	10	-200000	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	500	125	200	False	False	False	False
9	300000000	60	150	30	False	False	False	False
10	100000000	200	500000	10000	False	False	True	False

9.1 Calcolo del numero di features che superano la soglia per ogni riga

[5]:
```
df['Num_Outliers'] = df.filter(like='Outlier_').sum(axis=1)
df
```

[5]:

	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	200	4	10	-200000	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	500	125	200	False	False	False	False	0
9	300000000	60	150	30	False	False	False	False	0
10	100000000	200	500000	10000	False	False	True	False	1

9.2 Filtraggio dei dati per mantenere solo le righe con almeno il numero minimo di features superanti la soglia

[6]:
```
outliers = df[df['Num_Outliers'] >= min_features_threshold]
outliers
```

[6]:

	Feature1	Feature2	Feature3	Feature4	Outlier_Feature1	Outlier_Feature2	Outlier_Feature3	Outlier_Feature4	Num_Outliers
3	4	8	20000	4000000000	False	False	False	True	1
10	100000000	200	500000	10000	False	False	True	False	1

9.3 Identificazione e rimozione degli Outlier in un DataFrame

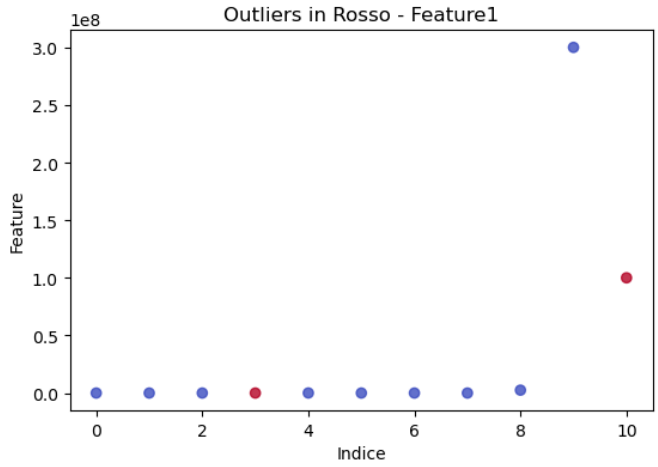
[7]:
```
# Aggiunge una colonna che indica se il record è un outlier o meno
df['Is_Outlier'] = df.index.isin(outliers.index)
# Rimuovi colonne ausiliarie
df.drop(df.filter(like='Outlier_').columns, axis=1, inplace=True)
df.drop('Num_Outliers', axis=1, inplace=True)
df
```

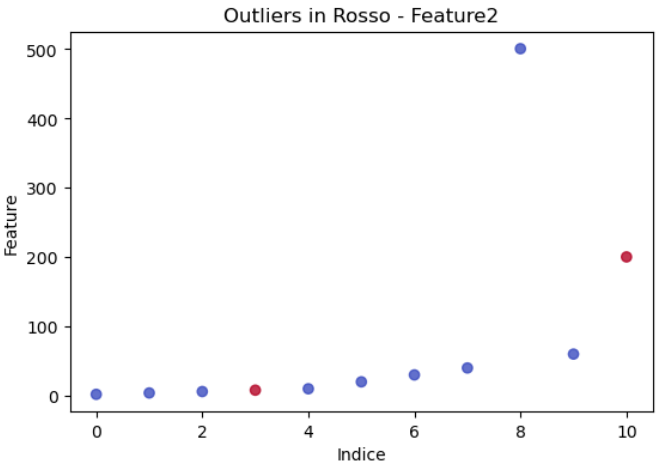
[7]:

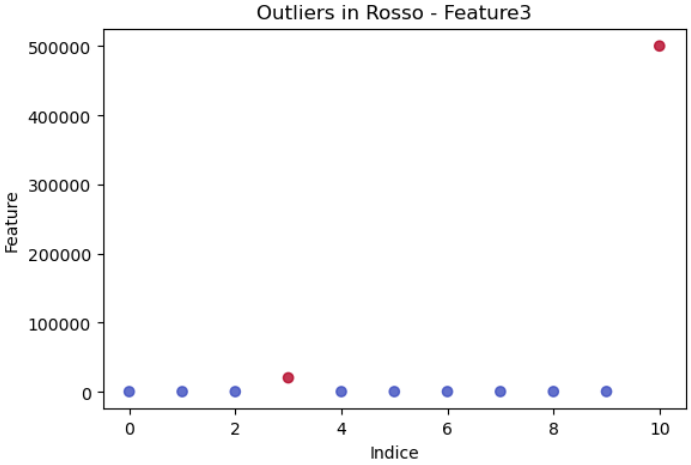
	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	200	4	10	-200000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	True
4	50000	10	25	5	False
5	10	20	50	10	False
6	15	30	75	15	False
7	20	40	100	20	False
8	2500000	500	125	200	False
9	300000000	60	150	30	False
10	100000000	200	500000	10000	True

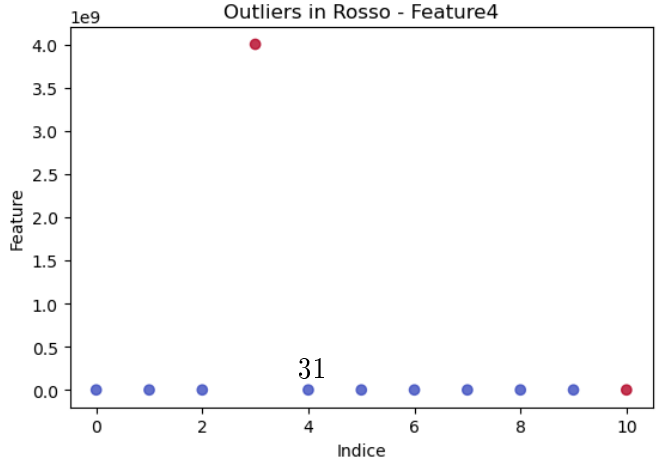
9.4 Visualizzazione Matrice dei Grafici con indicazione degli Outlier

[8]:
```
# 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_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
500
400
300
200
100
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

9.5 Eliminazione di righe che hanno una riga fuori scala

[9]:
```
# Elimina le righe corrispondenti agli outliers quelli che hanno almeno una␣
 ↪features fuoriscala
df_filtered = df[df['Is_Outlier'] == False]
df_filtered
```

[9]:

	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	200	4	10	-200000	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
8	2500000	500	125	200	False
9	300000000	60	150	30	False

9.6 Deviazione standard

[10]:
```
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, 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
```

10 Extra

---

11 Scarto interquartile (IQR)

11.1 Calcolo dell'IQR per un array

[11]:
```
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
```

11.2 Calcolo dell'IQR per una colonna di un Dataframe

[12]:
```
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
```

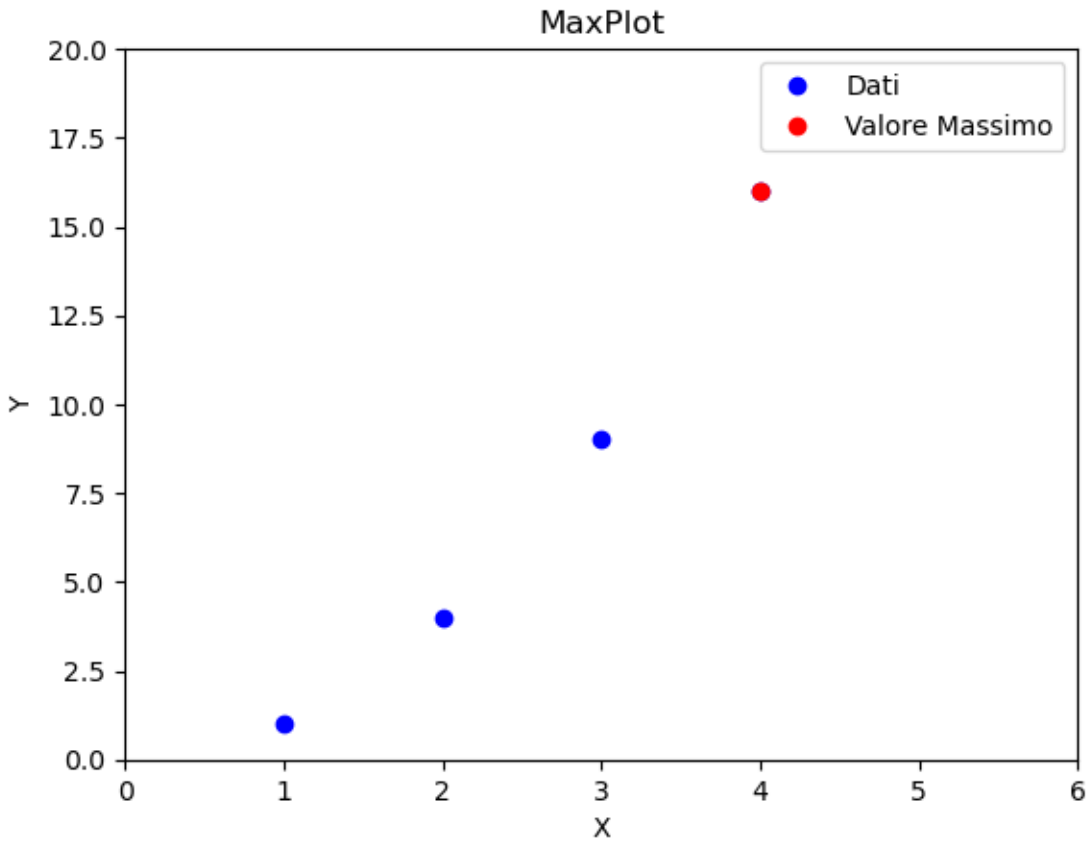
11.3 Calcolo dell'IQR per più colonne di un DataFrame

[13]:
```
# 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
```

12 Maxplot

[14]:
```
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()
```


MaxPlot
Dati
Valore Massimo
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Y
0
1
2
3
4
5
6
X