Secondo quadrimestre

Componenti del gruppo:

Justin Cadena
Francesco Miraglia
Zhou Zencheng

Indice:

1 Missing Values **4**
1.1 Creazione di un DataFrame con dati mancanti rappresentati da Nan e None 4
1.2 Creazione di un DataFrame con punteggi 5

2 Creazione di un DataFrame con dati mancanti e calcolo delle statistiche **6**
2.1 Selezione delle colonne numeriche 6
2.2 Sostituzione dei valori mancanti con le medie delle colonne numeriche 6
2.3 Sostituzione dei valori mancanti con le mode delle colonne categoriche 7
2.4 Stampa dei DataFrame: Primo con i valori mancanti e secondo con i valori mancanti sostituiti 7
2.5 Creazione di un DataFrame con Dati di Esempio 7
2.6 Conteggio dei valori mancanti per ciascuna colonna nel dataFrame 8
2.7 Percentuale di valori mancanti per ciascuna colonna nel dataFrame 8
2.8 Grafico del Dataframe 8
2.9 Funzione per il trattamento dei valori mancanti in variabili numeriche e categoriche . 9
2.10 Individua i valori mancanti 10
2.11 Creazione di una matrice di valori Mancanti in un DataFrame 11
2.12 Grafico della matrice 11
2.13 Informazione e descrizione del Dataframe 13
2.14 Valori mancanti per ciascuna colonna 14
2.15 Grafico matrice di correlazione 14
2.16 Generazione di un DataFrame con dati Casuali e introduzione di valori mancanti casuali 15
2.17 Conteggio delle righe con dati mancanti 16
2.18 Percentuale di valori mancanti di ogni colonna del Dataframe 16
2.19 Grafico della variabile 17
2.20 Grafico Matrice 17
2.21 Eliminazione delle righe mancanti in CatCol1 e CatCol2 18
2.22 Eliminazione delle righe mancanti di NumCol1, NumCol2 e NumCol3 19
2.23 Trattamento dei valori mancanti: Sostituzione con la moda per le Colonne Categoriche e media condizionata per le Colonne Numeriche 19
2.24 Generatore di dati casuali con sustituzione di moda e mediana 20

3 Import dati **21**
3.1 Visione generale del dataframe . . . 22
3.2 Filtraggio dei dati . . . 23

4 Splitting Dataset **27**

5 Train test **27**
5.1 Generazione e Suddivisione dei Dati . . . 29
5.2 Analisi della Relazione tra Visite di un Sito e Importo delle Vendite . . . 29
5.3 Analisi dei Dati di Fitness: Relazione tra Mesi Trascorsi e Peso Corporeo . . . 30
5.4 Confronto delle Distribuzioni di Età . . . 32
5.5 Split Stratificato dei Dati per il Training set e il Test set . . . 33
5.6 Grafico di distribuzione delle Classi nel Set . . . 33
5.7 Grafico di distribuzione delle Classi nel Training Set . . . 34
5.8 Grafico di distribuzione delle Classi nel Test Set . . . 35
5.9 Analisi Statistica su Campione Casuale e Dataset . . . 36
5.10 Creazione di DataFrame con Distribuzione Controllata . . . 37
5.11 Creazione di Subset di Dimensioni Simili da un DataFrame . . . 38
5.12 Analisi delle Distribuzioni delle Classi nei Subset . . . 38
5.13 Divisione dei Subset in Training Set e Test Set con Analisi delle Distribuzioni delle Classi . . . 40
5.14 Divisione dei Subset Stratificati in Training Set e Test Set con Analisi delle Distribuzioni delle Classi . . . 42
5.15 Analisi delle Distribuzioni delle Classi nei Subset con Divisione in Training Set e Test Set . . . 45

6 Extra **48**

7 Cross-validation **48**

8 Stratified Sampling (Campionamento stratificato) **48**

9 Time Series Split **49**

10 Outliers **49**
10.1 Rilevazione degli Outliers in un Dataframe . . . 50
10.2 Grafico a Dispersione . . . 50

11 Metodi per rilevare gli Outliers **51**

12 Z-score **51**
12.1 Calcolo del numero di features che superano la soglia per ogni riga . . . 53
12.2 Filtraggio dei dati per mantenere solo le righe con almeno il numero minimo di features superanti la soglia . . . 53
12.3 Identificazione e rimozione degli Outlier in un DataFrame . . . 54
12.4 Visualizzazione Matrice dei Grafici con indicazione degli Outlier . . . 54
12.5 Eliminazione di righe che hanno una riga fuori scala . . . 56
12.6 Deviazione standard . . . 56

13 Extra **57**

14 Scarto interquartile (IQR) **57**
14.1 Calcolo dell'IQR per un array 57
14.2 Calcolo dell'IQR per una colonna di un Dataframe 57
14.3 Calcolo dell'IQR per più colonne di un DataFrame 58

15 Maxplot **58**

1 Missing Values

1.1 Creazione di un DataFrame con dati mancanti rappresentati da Nan e None

[2]:
```
import pandas as pd

dataset = [
    {"età": 25, "punteggio": 90, "ammesso": 1},
    {"età": None, "punteggio": 85, "ammesso": 0},
    {"età": 28, "punteggio": None, "ammesso": 1},
    {"età": None, "punteggio": 75, "ammesso": 1},
    {"età": 23, "punteggio": None, "ammesso": None},
    {"età": 23, "punteggio": 77, "ammesso": None},
]
df = pd.DataFrame(dataset)
df
```

[2]:

	età	punteggio	ammesso
0	25.0	90.0	1.0
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

[3]:
```
#Seleziona solo le colonne 'punteggio' e 'ammesso' dal DataFrame
df[["punteggio", "ammesso"]]
```

[3]:

	punteggio	ammesso
0	90.0	1.0
1	85.0	0.0
2	NaN	1.0
3	75.0	1.0
4	NaN	NaN
5	77.0	NaN

[4]:
```
# Trova le righe contenenti dati mancanti nel DataFrame
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
righe_con_dati_mancanti
```

[4]:

	età	punteggio	ammesso
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

[5]:
```
# Calcola il totale delle righe con dati mancanti nel DataFrame
totale_dati_mancanti = righe_con_dati_mancanti.shape[0]
totale_dati_mancanti
```

[5]: 5

[6]:
```
# Stampa le righe con dati mancanti e il totale dei dati mancanti
print("Righe con dati mancanti:")
print(righe_con_dati_mancanti)
print("Totale dati mancanti:", totale_dati_mancanti)
```

```
Righe con dati mancanti:
    età  punteggio  ammesso
1   NaN       85.0      0.0
2  28.0        NaN      1.0
3   NaN       75.0      1.0
4  23.0        NaN      NaN
5  23.0       77.0      NaN
Totale dati mancanti: 5
```

1.2 Creazione di un DataFrame con punteggi

[7]:
```
import pandas as pd

dataset = [
    {"nome": "Alice", "età": 25, "punteggio": 90, "email": "alice@email.com"},
    {"nome": "Bob", "età": 22, "punteggio": None, "email": None},
    {"nome": "Charlie", "età": 28, "punteggio": 75, "email": "charlie@email.
 ↪com"},
]

# Eliminazione delle righe con dati mancanti
df1=df.dropna(inplace=False)
df1
```

[7]:

	età	punteggio	ammesso
0	25.0	90.0	1.0

[8]:
```
# Elimina le righe con dati mancanti dal DataFrame originale
df.dropna(inplace=True)
df
```

[8]:

	età	punteggio	ammesso
0	25.0	90.0	1.0

[9]:
```
# Elimina le righe con dati mancanti dal DataFrame originale
df.dropna(inplace=True)
df
```

[9]:
```
   età  punteggio  ammesso
0  25.0       90.0      1.0
```

2 Creazione di un DataFrame con dati mancanti e calcolo delle statistiche

[10]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

data = {
    'Variable1': [1, 2, 3, 4, 5],
    'Variable2': [1, 2, np.nan, 4, np.nan],
    'Missing_Column': ['A', 'B', 'A', 'C', np.nan]
}

df = pd.DataFrame(data)
df1=pd.DataFrame()
df
```

[10]:

	Variable1	Variable2	Missing_Column
0	1	1.0	A
1	2	2.0	B
2	3	NaN	A
3	4	4.0	C
4	5	NaN	NaN

2.1 Selezione delle colonne numeriche

[11]:
```
numeric_cols = df.select_dtypes(include=['number'])
numeric_cols.columns
```

[11]:
```
Index(['Variable1', 'Variable2'], dtype='object')
```

2.2 Sostituzione dei valori mancanti con le medie delle colonne numeriche

[12]:
```
df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.
↪columns].mean())
df1
```

[12]:

	Variable1	Variable2
0	1	1.000000
1	2	2.000000
2	3	2.333333
3	4	4.000000
4	5	2.333333

[13]:
```
categorical_cols = df.select_dtypes(exclude=['number'])
categorical_cols.columns
```

[13]:
```
Index(['Missing_Column'], dtype='object')
```

2.3 Sostituzione dei valori mancanti con le mode delle colonne categoriche

[14]:
```
df1[categorical_cols.columns] = df[categorical_cols.columns].
↪fillna(df[categorical_cols.columns].mode().iloc[0])
df1
```

[14]:

	Variable1	Variable2	Missing_Column
0	1	1.000000	A
1	2	2.000000	B
2	3	2.333333	A
3	4	4.000000	C
4	5	2.333333	A

2.4 Stampa dei DataFrame: Primo con i valori mancanti e secondo con i valori mancanti sostituiti

[15]:
```
print(f"il primo con i valori mancanti \n{df} \ne il secondo con i missing␣
↪values sostituiti \n{df1}")
```

```
il primo con i valori mancanti
   Variable1  Variable2 Missing_Column
0          1        1.0              A
1          2        2.0              B
2          3        NaN              A
3          4        4.0              C
4          5        NaN            NaN
e il secondo con i missing values sostituiti
   Variable1  Variable2 Missing_Column
0          1   1.000000              A
1          2   2.000000              B
2          3   2.333333              A
3          4   4.000000              C
4          5   2.333333              A
```

2.5 Creazione di un DataFrame con Dati di Esempio

---

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

# Genera dati di esempio
```

```
data = {
    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}
# Crea un DataFrame
df = pd.DataFrame(data)
df
```

[16]:

	Feature1	Feature2	Feature3
0	1.0	NaN	1.0
1	2.0	2.0	NaN
2	NaN	3.0	3.0
3	4.0	4.0	4.0
4	5.0	NaN	5.0

2.6 Conteggio dei valori mancanti per ciascuna colonna nel dataFrame

[17]:
```
df.isnull().sum()
```

[17]:
```
Feature1    1
Feature2    2
Feature3    1
dtype: int64
```

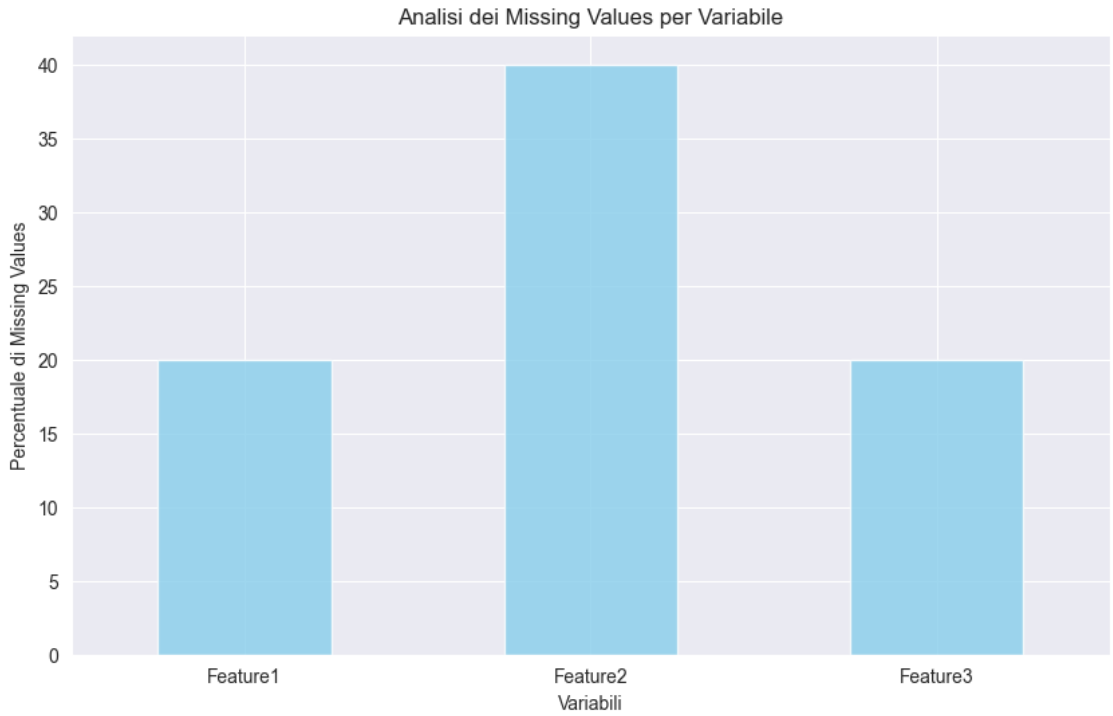
2.7 Percentuale di valori mancanti per ciascuna colonna nel dataFrame

[18]:
```
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent
```

[18]:
```
Feature1    20.0
Feature2    40.0
Feature3    20.0
dtype: float64
```

2.8 Grafico del Dataframe

[19]:
```
plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue',alpha=0.8)
plt.xlabel('Variabili')
plt.ylabel('Percentuale di Missing Values')
plt.title('Analisi dei Missing Values per Variabile')
plt.xticks(rotation=0)
plt.show()
```



2.9 Funzione per il trattamento dei valori mancanti in variabili numeriche e categoriche

[20]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Genera dati di esempio
data = {
    'Variable1': [1, 2, 3, 4, 5],
    'Variable2': [1, 2, np.nan, 4, np.nan],
    'Missing_Column': ['A', 'B', 'A', 'C', np.nan]
}

# Crea un DataFrame
df = pd.DataFrame(data)
df1=pd.DataFrame()


def missingvalues_sub(df):
    # Trattamento dei missing values nelle variabili numeriche e categoriche
    numeric_cols = df.select_dtypes(include=['number'])
```

```
    categorical_cols = df.select_dtypes(exclude=['number'])
    df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.
 ↪columns].mean())
    df1[categorical_cols.columns] = df[categorical_cols.columns].
 ↪fillna(df[categorical_cols.columns].mode().iloc[0])
    return df1


def main ():
    df1=missingvalues_sub(df)
    print(f"il primo con i valori mancanti \n{df} \ne il secondo con i missing␣
 ↪values sostituiti \n{df1}")


if __name__ == "__main__":
    main()
```

```
il primo con i valori mancanti
   Variable1  Variable2 Missing_Column
0          1        1.0              A
1          2        2.0              B
2          3        NaN              A
3          4        4.0              C
4          5        NaN            NaN
e il secondo con i missing values sostituiti
   Variable1  Variable2 Missing_Column
0          1   1.000000              A
1          2   2.000000              B
2          3   2.333333              A
3          4   4.000000              C
4          5   2.333333              A
```

2.10 Individua i valori mancanti

[21]:
```
df.isnull()
```

[21]:

	Variable1	Variable2	Missing_Column
0	False	False	False
1	False	False	False
2	False	True	False
3	False	False	False
4	False	True	True

2.11 Creazione di una matrice di valori Mancanti in un DataFrame

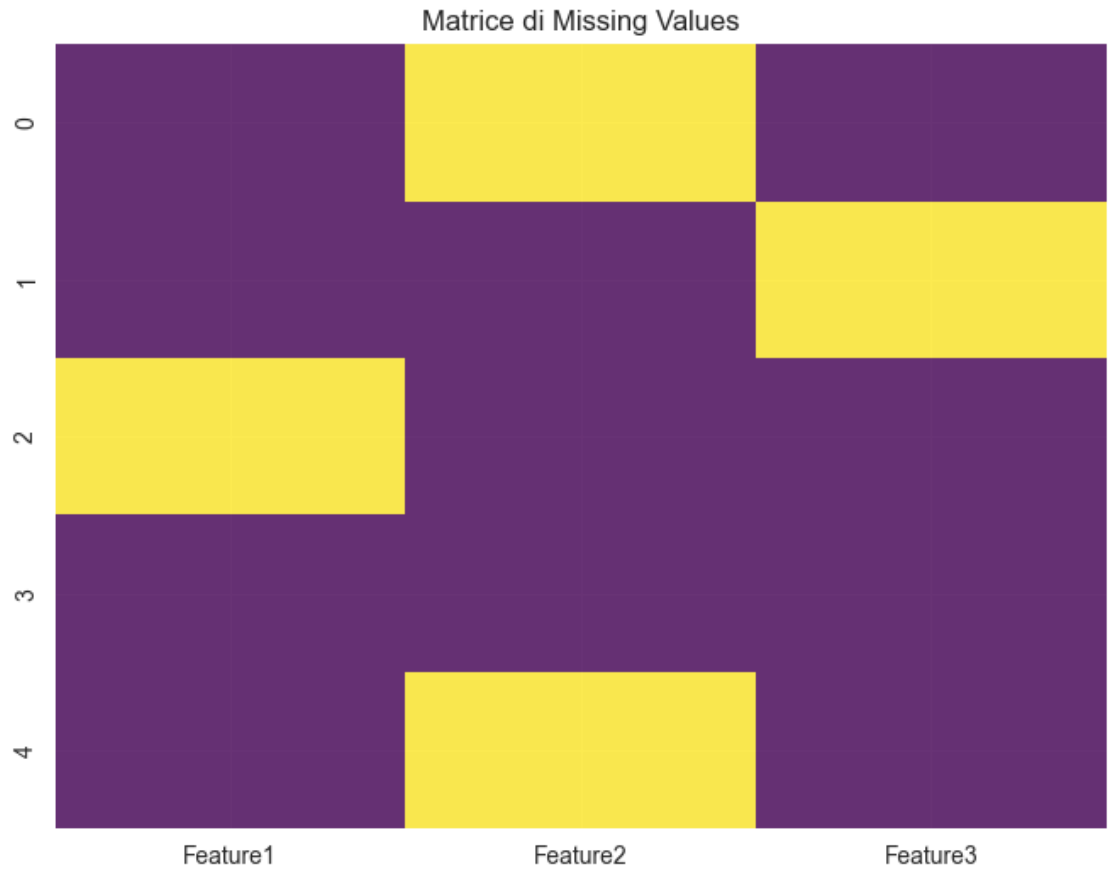
[22]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Genera dati di esempio
data = {
    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}

# Crea un DataFrame
df = pd.DataFrame(data)

# Calcola la matrice di missing values
missing_matrix = df.isnull()
missing_matrix
```

[22]:

	Feature1	Feature2	Feature3
0	False	True	False
1	False	False	True
2	True	False	False
3	False	False	False
4	False	True	False

2.12 Grafico della matrice

[23]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(missing_matrix, cmap='viridis', cbar=False,alpha=0.8)
plt.title('Matrice di Missing Values')
plt.show()
```



[24]:
```
## Creazione di un DataFrame con dati Casuali su Età, Genere, Punteggio e Reddito
```

[25]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Genera dati casuali per l'+
np.random.seed(2)
data = {
    'Età': np.random.randint(18, 70, size=1000),
    'Genere': np.random.choice(['Maschio', 'Femmina'], size=1000),
    'Punteggio': np.random.uniform(0, 100, size=1000),
    'Reddito': np.random.normal(50000, 15000, size=1000)
}

df = pd.DataFrame(data)
```

```
# Visualizza le prime righe del dataset
print(df.head(21))
```

```
    Età   Genere  Punteggio       Reddito
0    58  Maschio  93.309731  55174.034340
1    33  Femmina  97.279382  65873.059029
2    63  Femmina  91.185842  63246.553249
3    26  Femmina  75.926276  44534.875858
4    40  Maschio  25.156395  73444.267270
5    61  Femmina  90.055564  48451.939402
6    36  Femmina  29.717079  44579.517216
7    29  Femmina  87.762886  74639.606864
8    58  Femmina   4.139801  84279.892767
9    25  Femmina   5.641115  52083.863707
10   52  Maschio  80.315899  58188.649042
11   67  Maschio  10.670863  40301.012748
12   49  Maschio  43.920719  58292.619116
13   29  Femmina  34.315554  54842.947703
14   39  Maschio  27.790752  53270.120207
15   65  Maschio  36.205126  78821.228153
16   49  Maschio  48.566180  59639.075018
17   44  Femmina  83.643168  39339.223303
18   38  Maschio  61.718371  40687.283872
19   55  Femmina  90.736827  66795.123408
20   57  Maschio  83.670954  66695.930851
```

2.13 Informazione e descrizione del Dataframe

[26]:
```
print(df.info())

print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Età        1000 non-null   int32
 1   Genere     1000 non-null   object
 2   Punteggio  1000 non-null   float64
 3   Reddito    1000 non-null   float64
dtypes: float64(2), int32(1), object(1)
memory usage: 27.5+ KB
None
               Età    Punteggio       Reddito
count  1000.000000  1000.000000   1000.000000
mean     44.205000    48.687071  50036.084395
std      14.986847    29.617200  15027.142896
```

```
min      18.000000     0.090182   6017.070033
25%      31.000000    22.373740  39577.758808
50%      44.000000    47.030664  50994.854630
75%      58.000000    75.439618  60933.234680
max      69.000000    99.713537  96435.848804
```

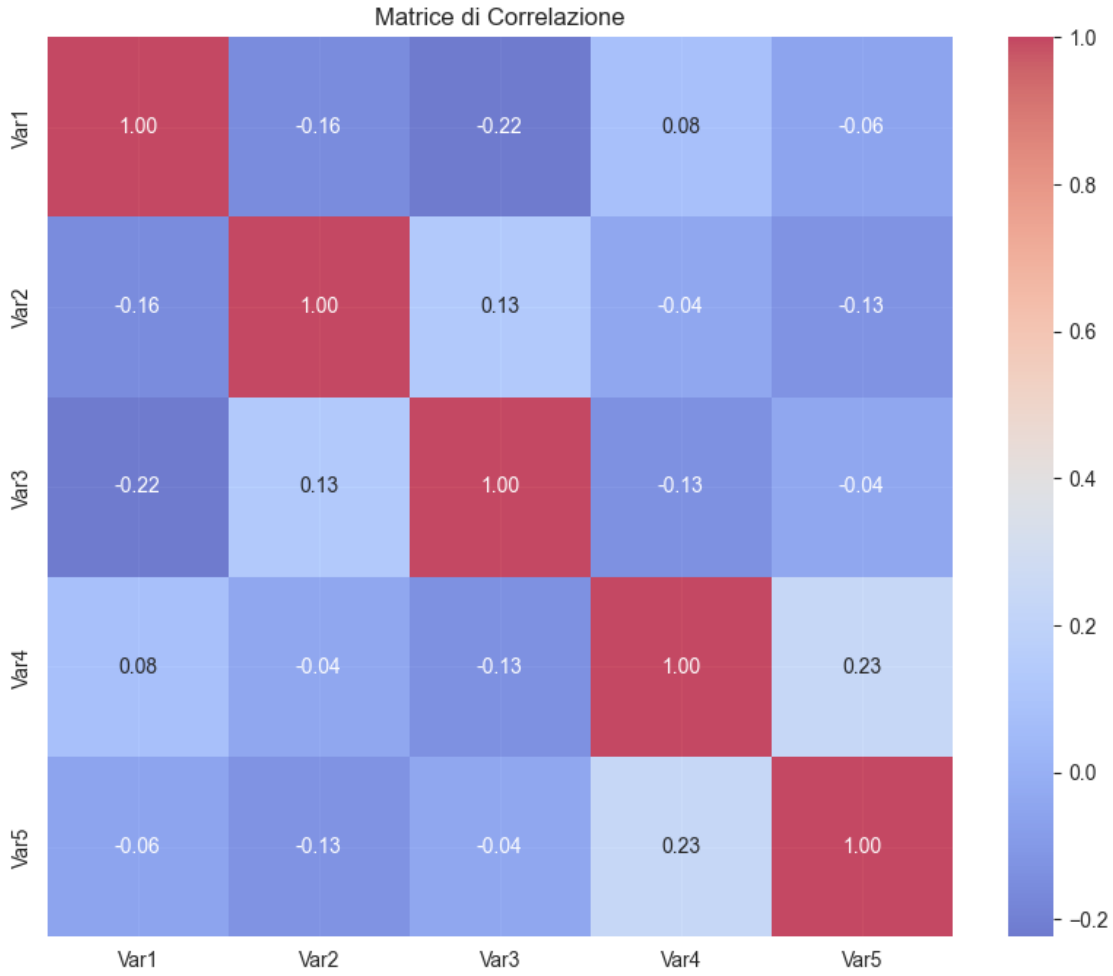
2.14 Valori mancanti per ciascuna colonna

[27]:
```
missing_data = df.isnull().sum()
print("Valori mancanti per ciascuna colonna:")
print(missing_data)
```

```
Valori mancanti per ciascuna colonna:
Età          0
Genere       0
Punteggio    0
Reddito      0
dtype: int64
```

2.15 Grafico matrice di correlazione

[28]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Genera un dataset di esempio con variabili numeriche
np.random.seed(42)
data = pd.DataFrame(np.random.rand(100, 5), columns=['Var1', 'Var2', 'Var3',␣
 ↪'Var4', 'Var5'])

# Aggiungi alcune variabili categoriche generate casualmente
data['Categoria1'] = np.random.choice(['A', 'B', 'C'], size=100)
data['Categoria2'] = np.random.choice(['X', 'Y'], size=100)

# Calcola la matrice di correlazione tra tutte le variabili numeriche
correlation_matrix = data.corr(numeric_only=True)

# Visualizza la matrice di correlazione come heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",alpha=0.7)
plt.title("Matrice di Correlazione")
plt.show()
```



2.16 Generazione di un DataFrame con dati Casuali e introduzione di valori mancanti casuali

[29]:
```
import pandas as pd
import numpy as np

# Impostare il seed per rendere i risultati riproducibili
np.random.seed(41)

# Creare un dataframe vuoto
df = pd.DataFrame()

# Generare dati casuali
n_rows = 10000
df['CatCol1'] = np.random.choice(['A', 'B', 'C'], size=n_rows)
df['CatCol2'] = np.random.choice(['X', 'Y'], size=n_rows)
```

```
df['NumCol1'] = np.random.randn(n_rows)
df['NumCol2'] = np.random.randint(1, 100, size=n_rows)
df['NumCol3'] = np.random.uniform(0, 1, size=n_rows)

# Calcolare il numero totale di missing values desiderati
total_missing_values = int(0.03 * n_rows * len(df.columns))

# Introdurre missing values casuali
for column in df.columns:
    num_missing_values = np.random.randint(0, total_missing_values + 1)
    missing_indices = np.random.choice(n_rows, size=num_missing_values,␣
 ↪replace=False)
    df.loc[missing_indices, column] = np.nan
df
```

[29]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
1	A	Y	1.945879	28.0	0.936825
2	C	X	0.988834	42.0	0.751516
3	A	Y	-0.181978	73.0	0.950696
4	B	X	2.080615	74.0	0.903045
...	...	...	...	...	...
9995	C	Y	1.352114	61.0	0.728445
9996	C	Y	1.143642	67.0	0.605930
9997	A	X	-0.665794	54.0	0.071041
9998	C	Y	0.004278	NaN	NaN
9999	A	X	0.622473	95.0	0.751384

[10000 rows x 5 columns]

2.17 Conteggio delle righe con dati mancanti

[30]:
```
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
len(righe_con_dati_mancanti)
```

[30]: 3648

2.18 Percentuale di valori mancanti di ogni colonna del Dataframe

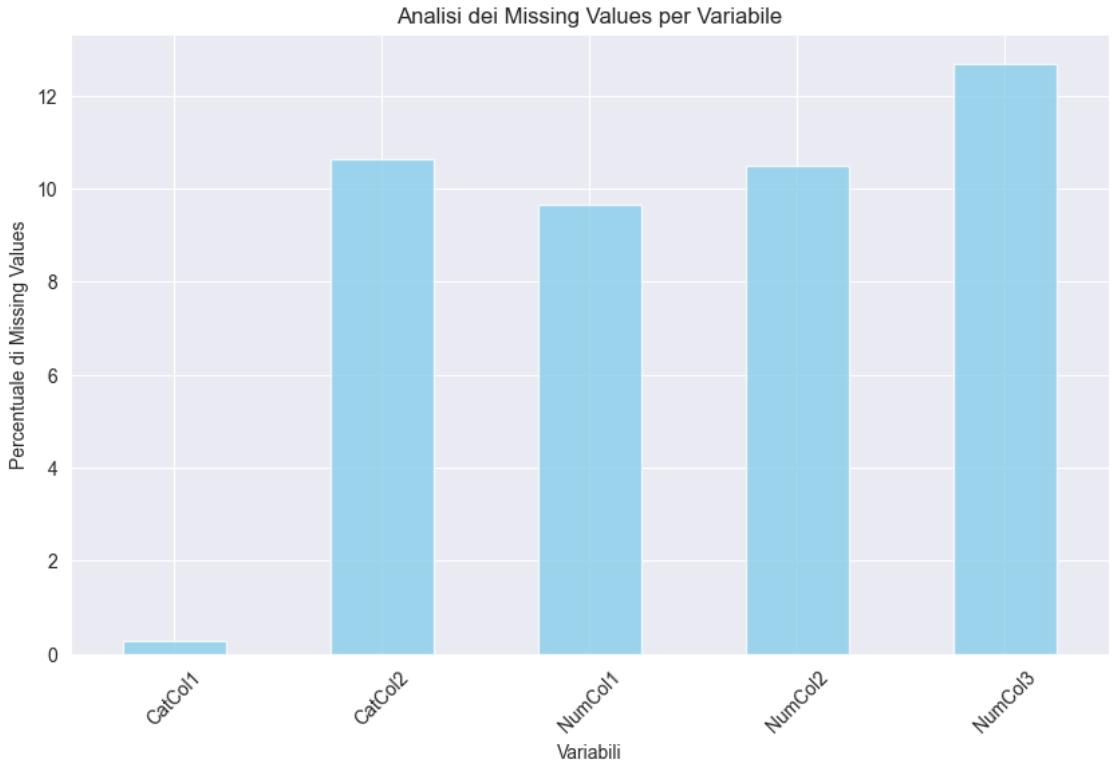
[31]:
```
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent
```

[31]:
```
CatCol1     0.29
CatCol2    10.63
NumCol1     9.67
NumCol2    10.48
NumCol3    12.69
```

```
dtype: float64
```

2.19 Grafico della variabile

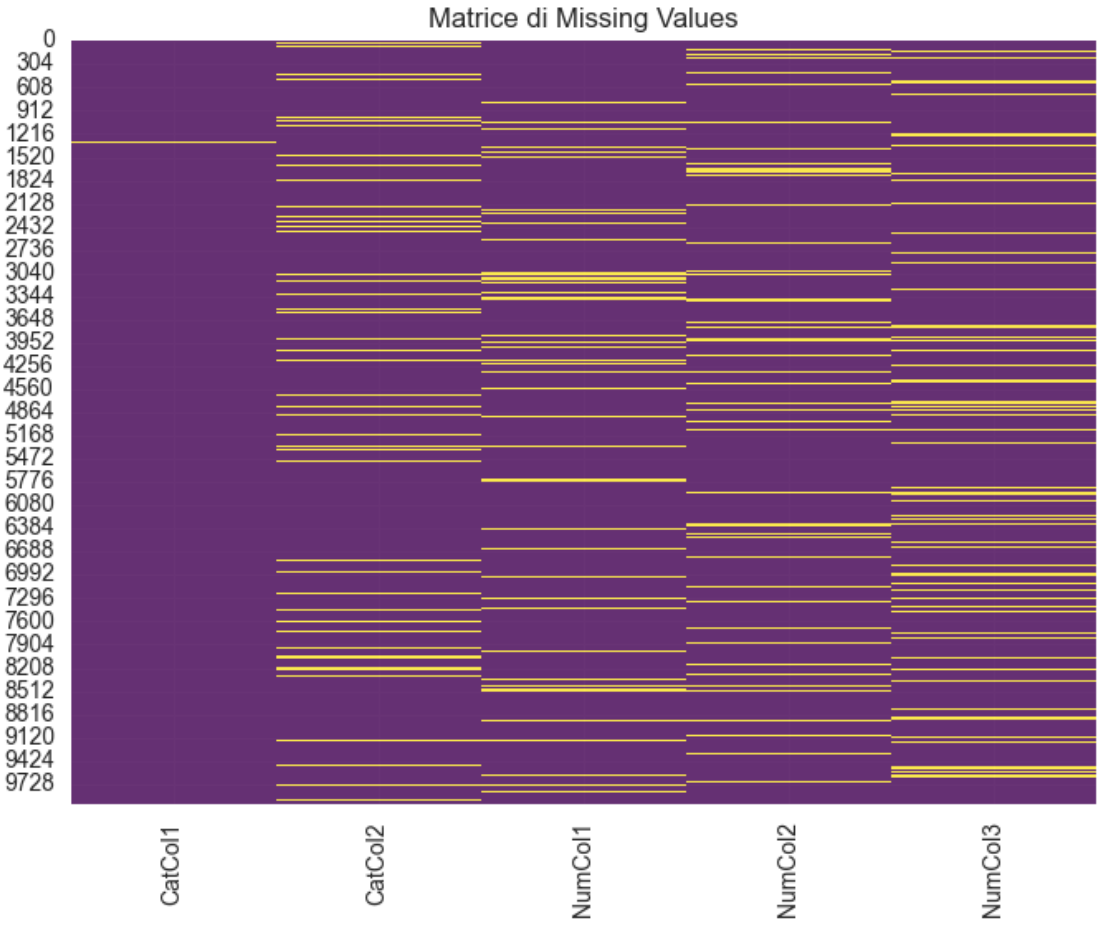
[32]:
```
plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue',alpha=0.8)
plt.xlabel('Variabili')
plt.ylabel('Percentuale di Missing Values')
plt.title('Analisi dei Missing Values per Variabile')
plt.xticks(rotation=45)
plt.show()
```



2.20 Grafico Matrice

[33]:
```
missing_matrix = df.isnull()

# Crea una heatmap colorata
plt.figure(figsize=(8, 6))
sns.heatmap(missing_matrix, cmap='viridis', cbar=False,alpha=0.8)
plt.title('Matrice di Missing Values')
plt.xticks(rotation=90)
```

```
plt.show()
```



2.21 Eliminazione delle righe mancanti in CatCol1 e CatCol2

[34]:
```
df = df.dropna(subset=["CatCol1", 'CatCol2'], how='all')
df
```

[34]:

	CatCol1	CatCol2	NumCol1	NumCol2	NumCol3
0	A	NaN	0.440877	49.0	0.246007
1	A	Y	1.945879	28.0	0.936825
2	C	X	0.988834	42.0	0.751516
3	A	Y	-0.181978	73.0	0.950696
4	B	X	2.080615	74.0	0.903045
...	...	...	...	...	...
9995	C	Y	1.352114	61.0	0.728445
9996	C	Y	1.143642	67.0	0.605930
9997	A	X	-0.665794	54.0	0.071041

```
9998       C       Y  0.004278      NaN       NaN
9999       A       X  0.622473     95.0  0.751384

[9995 rows x 5 columns]
```

2.22 Eliminazione delle righe mancanti di NumCol1, NumCol2 e NumCol3

[35]:
```
df = df.dropna(subset=["NumCol1", "NumCol2","NumCol3"], how='all')
df
```

[35]:
```
     CatCol1 CatCol2   NumCol1  NumCol2   NumCol3
0          A     NaN  0.440877     49.0  0.246007
1          A       Y  1.945879     28.0  0.936825
2          C       X  0.988834     42.0  0.751516
3          A       Y -0.181978     73.0  0.950696
4          B       X  2.080615     74.0  0.903045
...      ...     ...       ...      ...       ...
9995       C       Y  1.352114     61.0  0.728445
9996       C       Y  1.143642     67.0  0.605930
9997       A       X -0.665794     54.0  0.071041
9998       C       Y  0.004278      NaN       NaN
9999       A       X  0.622473     95.0  0.751384

[9975 rows x 5 columns]
```

2.23 Trattamento dei valori mancanti: Sostituzione con la moda per le Colonne Categoriche e media condizionata per le Colonne Numeriche

[36]:
```
numeric_cols = df.select_dtypes(include=['number'])
categorical_cols = df.select_dtypes(exclude=['number'])

# Copia esplicità del DataFrame per evitare modifiche indesiderate
df_copy = df.copy()

# Sostituisci i missing values nelle colonne categoriche con la moda
df_copy.loc[:, categorical_cols.columns] = df_copy[categorical_cols.columns].
 ↪fillna(df_copy[categorical_cols.columns].mode().iloc[0])

# Calcola la media condizionata solo per le colonne numeriche con dati mancanti
conditional_means = df_copy.groupby('CatCol1')[numeric_cols.columns].
 ↪transform('mean')

# Aggiorna solo i valori mancanti nelle colonne numeriche con la media␣
 ↪condizionata
df_copy[numeric_cols.columns] = df_copy[numeric_cols.columns].
 ↪fillna(conditional_means)
```

```
# Stampare il DataFrame risultante
print(df_copy)
```

```
     CatCol1 CatCol2   NumCol1    NumCol2   NumCol3
0          A       Y  0.440877  49.000000  0.246007
1          A       Y  1.945879  28.000000  0.936825
2          C       X  0.988834  42.000000  0.751516
3          A       Y -0.181978  73.000000  0.950696
4          B       X  2.080615  74.000000  0.903045
...      ...     ...       ...        ...       ...
9995       C       Y  1.352114  61.000000  0.728445
9996       C       Y  1.143642  67.000000  0.605930
9997       A       X -0.665794  54.000000  0.071041
9998       C       Y  0.004278  49.845018  0.489352
9999       A       X  0.622473  95.000000  0.751384

[9975 rows x 5 columns]
```

2.24 Generatore di dati casuali con sustituzione di moda e mediana

[37]:
```
import pandas as pd
import numpy as np

# Impostare il seed per rendere i risultati riproducibili
np.random.seed(41)

# Creare un dataframe vuoto
df = pd.DataFrame()

# Generare dati casuali
n_rows = 10000000
df['CatCol1'] = np.random.choice(['A', 'B', 'C'], size=n_rows)
df['CatCol2'] = np.random.choice(['X', 'Y'], size=n_rows)
df['NumCol1'] = np.random.randn(n_rows)
df['NumCol2'] = np.random.randint(1, 100, size=n_rows)
df['NumCol3'] = np.random.uniform(0, 1, size=n_rows)

# Calcolare il numero totale di missing values desiderati
total_missing_values = int(0.05 * n_rows * len(df.columns))

# Introdurre missing values casuali
for column in df.columns:
    num_missing_values = np.random.randint(0, total_missing_values + 1)
    missing_indices = np.random.choice(n_rows, size=num_missing_values,␣
 ↪replace=False)
    df.loc[missing_indices, column] = np.nan

# Elimina le righe in cui entrambe le features categoriche hanno valori NaN
```

```
df = df.dropna(subset=["CatCol1", 'CatCol2'], how='all')
df = df.dropna(subset=["NumCol1", 'NumCol2','NumCol3'], how='all')


numeric_cols = df.select_dtypes(include=['number'])
categorical_cols = df.select_dtypes(exclude=['number'])

# Sostituisci i missing values nelle colonne categoriche con la moda utilizzando␣
 ↪.loc
df.loc[:, categorical_cols.columns] = df[categorical_cols.columns].
 ↪fillna(df[categorical_cols.columns].mode().iloc[0])

# Calcola la media condizionata solo per le colonne numeriche con dati mancanti
conditional_means = df[numeric_cols.columns].fillna(df.
 ↪groupby('CatCol1')[numeric_cols.columns].transform('mean'))

# Aggiorna le colonne numeriche con la media condizionata utilizzando .loc
df.loc[:, numeric_cols.columns] = conditional_means

# Stampa il DataFrame risultante
print(df)
```

```
        CatCol1 CatCol2   NumCol1  NumCol2   NumCol3
0             A       Y -0.391604     98.0  0.409815
1             A       X  0.000551     19.0  0.886592
2             C       Y  1.266001     52.0  0.848556
3             A       X  0.449617     70.0  0.546525
4             B       X  0.742505     72.0  0.467257
...         ...     ...       ...      ...       ...
9999995       A       Y  0.464663      7.0  0.992815
9999996       A       X  0.149775     13.0  0.731368
9999997       C       Y -0.608376      1.0  0.606349
9999998       C       Y  0.000101     69.0  0.115812
9999999       B       Y  1.666715     76.0  0.245699

[9635330 rows x 5 columns]
```

3 Import dati

---

[38]:
```
import pandas as pd

# Legge il file CSV e lo salva in un dataframe
df1 = pd.read_csv('Pokemon.csv')

# Mostra le prime righe del dataframe per verificare l'importazione
```

```
df1.head()
```

[38]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

3.1 Visione generale del dataframe

[39]:
```
print(df1.info())
print(df1.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           800 non-null    int64
 1   Name        800 non-null    object
 2   Type 1      800 non-null    object
 3   Type 2      414 non-null    object
 4   Total       800 non-null    int64
 5   HP          800 non-null    int64
 6   Attack      800 non-null    int64
 7   Defense     800 non-null    int64
 8   Sp. Atk     800 non-null    int64
 9   Sp. Def     800 non-null    int64
 10  Speed       800 non-null    int64
 11  Generation  800 non-null    int64
 12  Legendary   800 non-null    bool
dtypes: bool(1), int64(9), object(3)
memory usage: 75.9+ KB
None
                #       Total          HP      Attack     Defense     Sp. Atk  \
count  800.000000  800.00000  800.000000  800.000000  800.000000  800.000000
mean   362.813750  435.10250   69.258750   79.001250   73.842500   72.820000
std    208.343798  119.96304   25.534669   32.457366   31.183501   32.722294
min      1.000000  180.00000    1.000000    5.000000    5.000000   10.000000
25%    184.750000  330.00000   50.000000   55.000000   50.000000   49.750000
```

```
50%    364.500000  450.00000   65.000000   75.000000   70.000000   65.000000
75%    539.250000  515.00000   80.000000  100.000000   90.000000   95.000000
max    721.000000  780.00000  255.000000  190.000000  230.000000  194.000000

          Sp. Def       Speed  Generation
count  800.000000  800.000000   800.00000
mean    71.902500   68.277500     3.32375
std     27.828916   29.060474     1.66129
min     20.000000    5.000000     1.00000
25%     50.000000   45.000000     2.00000
50%     70.000000   65.000000     3.00000
75%     90.000000   90.000000     5.00000
max    230.000000  180.000000     6.00000
```

3.2 Filtraggio dei dati

[40]:
```
# Seleziona una singola colonna
print(df1['Name'])

# Seleziona righe basate su condizioni
print(df1[df1['HP'] > 150])
```

```
0                 Bulbasaur
1                   Ivysaur
2                  Venusaur
3      VenusaurMega Venusaur
4                Charmander
              ...
795                 Diancie
796      DiancieMega Diancie
797      HoopaHoopa Confined
798       HoopaHoopa Unbound
799                Volcanion
Name: Name, Length: 800, dtype: object
       #       Name   Type 1 Type 2  Total   HP  Attack  Defense  Sp. Atk  \
121  113    Chansey   Normal    NaN    450  250       5        5       35
155  143    Snorlax   Normal    NaN    540  160     110       65       65
217  202  Wobbuffet  Psychic    NaN    405  190      33       58       33
261  242    Blissey   Normal    NaN    540  255      10       10       75
351  321    Wailord    Water    NaN    500  170      90       45       90
655  594  Alomomola    Water    NaN    470  165      75       80       40

     Sp. Def  Speed  Generation  Legendary
121      105     50           1      False
155      110     30           1      False
217       58     33           2      False
261      135     55           2      False
351       45     60           3      False
```

```
655       45     65           5      False
```

[41]:
```
import pandas as pd

df = pd.read_csv('Serie A.csv')

df.head()
```

[41]:
```
  Div        Date HomeTeam  AwayTeam  FTHG  FTAG FTR  HTHG  HTAG HTR  ...  \
0  I1  18/08/2018   Chievo  Juventus     2     3   A     1     1   D  ...
1  I1  18/08/2018    Lazio    Napoli     1     2   A     1     1   D  ...
2  I1  19/08/2018  Bologna      Spal     0     1   A     0     0   D  ...
3  I1  19/08/2018   Empoli  Cagliari     2     0   H     1     0   H  ...
4  I1  19/08/2018    Parma   Udinese     2     2   D     1     0   H  ...

   BbAv<2.5  BbAH  BbAHh  BbMxAHH  BbAvAHH  BbMxAHA  BbAvAHA   PSCH  PSCD  \
0      2.13    19   2.00     1.68     1.64     2.38     2.29  18.84  6.42
1      2.17    20   0.00     2.12     2.07     1.83     1.79   2.78  3.57
2      1.58    19  -0.25     1.97     1.92     1.99     1.94   2.31  3.18
3      1.71    19  -0.25     1.98     1.91     1.98     1.94   2.54  3.42
4      1.65    20   0.00     1.81     1.77     2.18     2.10   2.80  3.24

   PSCA
0  1.22
1  2.59
2  3.59
3  2.95
4  2.78

[5 rows x 61 columns]
```

[42]:
```
import csv

# Apre il file CSV e lo legge
with open('Serie A.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

```
['Div', 'Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR', 'HTHG', 'HTAG',
'HTR', 'HS', 'AS', 'HST', 'AST', 'HF', 'AF', 'HC', 'AC', 'HY', 'AY', 'HR', 'AR',
'B365H', 'B365D', 'B365A', 'BWH', 'BWD', 'BWA', 'IWH', 'IWD', 'IWA', 'PSH',
'PSD', 'PSA', 'WHH', 'WHD', 'WHA', 'VCH', 'VCD', 'VCA', 'Bb1X2', 'BbMxH',
'BbAvH', 'BbMxD', 'BbAvD', 'BbMxA', 'BbAvA', 'BbOU', 'BbMx>2.5', 'BbAv>2.5',
'BbMx<2.5', 'BbAv<2.5', 'BbAH', 'BbAHh', 'BbMxAHH', 'BbAvAHH', 'BbMxAHA',
'BbAvAHA', 'PSCH', 'PSCD', 'PSCA']
['I1', '18/08/2018', 'Chievo', 'Juventus', '2', '3', 'A', '1', '1', 'D', '7',
'23', '2', '11', '7', '9', '0', '8', '2', '0', '0', '0', '13', '5.75', '1.25',
```

'14', '6.25', '1.22', '13', '5.8', '1.25', '15.87', '6.21', '1.25', '13', '6',
'1.22', '15', '6', '1.25', '41', '15.87', '13.7', '6.44', '5.88', '1.27',
'1.24', '38', '1.75', '1.7', '2.24', '2.13', '19', '2', '1.68', '1.64', '2.38',
'2.29', '18.84', '6.42', '1.22']
['I1', '18/08/2018', 'Lazio', 'Napoli', '1', '2', 'A', '1', '1', 'D', '9', '11',
'5', '6', '8', '5', '4', '7', '0', '0', '0', '0', '2.8', '3.4', '2.5', '2.8',
'3.5', '2.45', '2.8', '3.55', '2.4', '2.9', '3.59', '2.48', '2.8', '3.5', '2.4',
'2.88', '3.4', '2.5', '41', '2.95', '2.82', '3.65', '3.5', '2.56', '2.44', '38',
'1.74', '1.68', '2.29', '2.17', '20', '0', '2.12', '2.07', '1.83', '1.79',
'2.78', '3.57', '2.59']
['I1', '19/08/2018', 'Bologna', 'Spal', '0', '1', 'A', '0', '0', 'D', '8', '10',
'3', '5', '16', '11', '7', '0', '4', '2', '1', '0', '2.25', '3.2', '3.4',
'2.25', '3.2', '3.4', '2.25', '3.1', '3.45', '2.32', '3.16', '3.58', '2.25',
'3.2', '3.3', '2.25', '3.1', '3.7', '41', '2.39', '2.27', '3.21', '3.12', '3.7',
'3.44', '38', '2.43', '2.34', '1.65', '1.58', '19', '-0.25', '1.97', '1.92',
'1.99', '1.94', '2.31', '3.18', '3.59']
['I1', '19/08/2018', 'Empoli', 'Cagliari', '2', '0', 'H', '1', '0', 'H', '9',
'12', '4', '5', '19', '19', '6', '6', '3', '3', '0', '0', '2.14', '3.2', '3.6',
'2.15', '3.25', '3.6', '2.15', '3.3', '3.45', '2.22', '3.26', '3.6', '2.25',
'3.25', '3.25', '2.25', '3.13', '3.6', '41', '2.34', '2.22', '3.35', '3.23',
'3.65', '3.43', '38', '2.22', '2.11', '1.78', '1.71', '19', '-0.25', '1.98',
'1.91', '1.98', '1.94', '2.54', '3.42', '2.95']
['I1', '19/08/2018', 'Parma', 'Udinese', '2', '2', 'D', '1', '0', 'H', '9',
'16', '6', '6', '10', '13', '4', '5', '2', '2', '0', '0', '2.45', '3.3', '2.9',
'2.45', '3.2', '3', '2.45', '3.15', '3.05', '2.51', '3.2', '3.18', '2.5', '3.2',
'2.9', '2.6', '3.1', '3', '41', '2.6', '2.47', '3.3', '3.17', '3.18', '3', '38',
'2.34', '2.21', '1.73', '1.65', '20', '0', '1.81', '1.77', '2.18', '2.1', '2.8',
'3.24', '2.78']
['I1', '19/08/2018', 'Sassuolo', 'Inter', '1', '0', 'H', '1', '0', 'H', '7',
'11', '5', '4', '23', '11', '4', '4', '3', '2', '0', '0', '5.5', '4', '1.61',
'5.25', '3.9', '1.65', '5.4', '4', '1.6', '5.96', '4.19', '1.61', '5', '3.8',
'1.65', '6.25', '3.9', '1.6', '41', '6.25', '5.45', '4.21', '3.96', '1.67',
'1.62', '38', '1.92', '1.83', '2.04', '1.96', '18', '1', '1.83', '1.79', '2.16',
'2.08', '5.08', '3.71', '1.79']
['I1', '19/08/2018', 'Torino', 'Roma', '0', '1', 'A', '0', '0', 'D', '8', '16',
'5', '13', '10', '14', '6', '5', '1', '3', '0', '0', '3.5', '3.5', '2.1', '3.5',
'3.6', '2.05', '3.4', '3.55', '2.1', '3.59', '3.59', '2.14', '3.4', '3.6',
'2.05', '3.6', '3.5', '2.1', '40', '3.65', '3.45', '3.67', '3.53', '2.19',
'2.09', '37', '1.79', '1.72', '2.17', '2.09', '19', '0.25', '2.11', '2.04',
'1.85', '1.81', '3.62', '3.55', '2.14']
['I1', '20/08/2018', 'Atalanta', 'Frosinone', '4', '0', 'H', '1', '0', 'H',
'16', '6', '10', '1', '14', '10', '4', '4', '0', '2', '0', '0', '1.36', '4.5',
'11', '1.36', '5', '8.5', '1.35', '4.95', '9.6', '1.35', '5.01', '10.76',
'1.35', '4.8', '9', '1.33', '5', '12', '40', '1.38', '1.35', '5.25', '4.84',
'12', '9.69', '38', '1.88', '1.81', '2.05', '1.98', '19', '-1.5', '2.14',
'2.06', '1.86', '1.81', '1.35', '5.26', '10.54']
['I1', '25/08/2018', 'Juventus', 'Lazio', '2', '0', 'H', '1', '0', 'H', '14',
'8', '9', '3', '12', '13', '6', '2', '3', '2', '0', '0', '1.36', '5', '8.5',

'1.36', '5', '8.5', '1.38', '4.95', '8', '1.36', '5.2', '10.07', '1.36', '4.75',
'8', '1.33', '5', '10', '41', '1.4', '1.36', '5.38', '5.01', '10.25', '8.73',
'38', '1.69', '1.63', '2.36', '2.25', '21', '-1', '1.6', '1.56', '2.6', '2.46',
'1.51', '4.53', '7.08']
['I1', '25/08/2018', 'Napoli', 'Milan', '3', '2', 'H', '0', '1', 'A', '21', '6',
'13', '4', '12', '9', '5', '1', '2', '2', '0', '0', '1.61', '4', '5.5', '1.62',
'4', '5.5', '1.67', '3.95', '5.1', '1.68', '3.92', '5.7', '1.65', '3.9', '5',
'1.65', '4', '5.5', '41', '1.71', '1.65', '4.2', '3.92', '5.78', '5.32', '38',
'1.83', '1.75', '2.16', '2.06', '22', '-1', '2.23', '2.14', '1.77', '1.73',
'1.65', '4.01', '5.94']
['I1', '26/08/2018', 'Cagliari', 'Sassuolo', '2', '2', 'D', '1', '0', 'H', '12',
'12', '7', '6', '14', '14', '7', '2', '4', '4', '0', '1', '2.45', '3.3', '2.9',
'2.55', '3.2', '2.9', '2.55', '3.25', '2.85', '2.61', '3.35', '2.93', '2.5',
'3.3', '2.8', '2.63', '3.2', '2.9', '41', '2.64', '2.54', '3.37', '3.23', '3',
'2.87', '38', '2.12', '2.05', '1.83', '1.75', '20', '0', '1.86', '1.82', '2.08',
'2.03', '2.48', '3.3', '3.13']
['I1', '26/08/2018', 'Fiorentina', 'Chievo', '6', '1', 'H', '2', '0', 'H', '19',
'5', '15', '3', '14', '8', '2', '3', '1', '3', '0', '0', '1.5', '4.33', '6.5',
'1.45', '4.4', '7.5', '1.5', '4.2', '6.7', '1.46', '4.52', '8.32', '1.44',
'4.4', '7', '1.45', '4.5', '7.5', '40', '1.5', '1.45', '4.66', '4.38', '8.32',
'7.38', '37', '1.89', '1.83', '2.05', '1.97', '22', '-1', '1.78', '1.74',
'2.22', '2.12', '1.45', '4.58', '8.36']
['I1', '26/08/2018', 'Frosinone', 'Bologna', '0', '0', 'D', '0', '0', 'D', '11',
'10', '7', '5', '15', '12', '2', '5', '2', '1', '0', '0', '3.3', '3', '2.4',
'3.1', '3.25', '2.4', '3.2', '3.15', '2.35', '3.2', '3.38', '2.4', '3.1', '3.3',
'2.3', '3.4', '3', '2.4', '41', '3.4', '3.18', '3.38', '3.18', '2.47', '2.37',
'38', '2.44', '2.32', '1.65', '1.59', '20', '0.25', '1.89', '1.85', '2.06',
'2.01', '3.54', '3.12', '2.37']
['I1', '26/08/2018', 'Genoa', 'Empoli', '2', '1', 'H', '2', '0', 'H', '10',
'12', '4', '8', '14', '15', '3', '6', '3', '2', '0', '0', '1.95', '3.4', '4',
'2', '3.3', '4', '2', '3.3', '4', '2.03', '3.36', '4.23', '2', '3.3', '3.9',
'2', '3.25', '4.33', '41', '2.07', '1.99', '3.45', '3.28', '4.33', '4.08', '38',
'2.42', '2.29', '1.67', '1.61', '19', '-0.25', '1.74', '1.71', '2.27', '2.19',
'2.02', '3.58', '3.96']
['I1', '26/08/2018', 'Inter', 'Torino', '2', '2', 'D', '2', '0', 'H', '9', '10',
'6', '7', '16', '12', '5', '4', '1', '2', '0', '0', '1.5', '4.33', '7', '1.48',
'4.4', '6.75', '1.5', '4.4', '6.4', '1.49', '4.66', '7.15', '1.44', '4.4', '7',
'1.5', '4.5', '6.5', '42', '1.53', '1.49', '4.74', '4.42', '7.15', '6.57', '39',
'1.79', '1.72', '2.21', '2.09', '23', '-1', '1.85', '1.81', '2.1', '2.05',
'1.59', '3.98', '6.91']
['I1', '26/08/2018', 'Spal', 'Parma', '1', '0', 'H', '0', '0', 'D', '15', '3',
'5', '0', '13', '14', '7', '2', '5', '2', '0', '0', '2.04', '3.3', '3.8', '2.1',
'3.2', '3.9', '2.05', '3.2', '3.95', '2.11', '3.22', '4.11', '2.05', '3.3',
'3.7', '2.1', '3.13', '4.2', '40', '2.15', '2.07', '3.35', '3.21', '4.2',
'3.93', '37', '2.42', '2.31', '1.66', '1.6', '19', '-0.25', '1.81', '1.77',
'2.16', '2.1', '2.47', '3.01', '3.47']
['I1', '26/08/2018', 'Udinese', 'Sampdoria', '1', '0', 'H', '1', '0', 'H', '13',
'9', '5', '4', '6', '10', '8', '7', '1', '3', '0', '0', '2.4', '3.25', '3.1',

```
'2.4', '3.2', '3.1', '2.35', '3.35', '3', '2.44', '3.45', '3.06', '2.38', '3.4',
'2.9', '2.38', '3.25', '3.2', '41', '2.47', '2.37', '3.45', '3.32', '3.2',
'3.04', '38', '1.97', '1.9', '1.98', '1.89', '20', '-0.25', '2.08', '2.04',
'1.87', '1.83', '2.27', '3.51', '3.34']
['I1', '27/08/2018', 'Roma', 'Atalanta', '3', '3', 'D', '1', '3', 'A', '21',
'17', '11', '10', '9', '13', '3', '11', '1', '1', '0', '0', '1.72', '3.75', '5',
'1.7', '4', '4.75', '1.7', '3.9', '4.9', '1.72', '4.13', '4.98', '1.7', '4',
'4.5', '1.7', '3.8', '5.25', '41', '1.76', '1.71', '4.13', '3.87', '5.25',
'4.84', '39', '1.9', '1.83', '2.03', '1.96', '22', '-1', '2.38', '2.28', '1.72',
'1.65', '1.57', '4.19', '6.57']
['I1', '31/08/2018', 'Milan', 'Roma', '2', '1', 'H', '1', '0', 'H', '18', '6',
'9', '3', '8', '6', '8', '3', '0', '2', '0', '0', '2.45', '3.4', '2.87', '2.4',
'3.4', '2.95', '2.4', '3.45', '2.9', '2.49', '3.48', '2.98', '2.45', '3.4',
'2.8', '2.45', '3.4', '3', '39', '2.6', '2.44', '3.57', '3.42', '3.06', '2.9',
'37', '1.81', '1.75', '2.17', '2.07', '20', '-0.25', '2.16', '2.12', '1.81',
'1.78', '2.41', '3.42', '3.14']
```

4 Splitting Dataset

---

5 Train test

[1]:
```python
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
```

5.1 Generazione e Suddivisione dei Dati

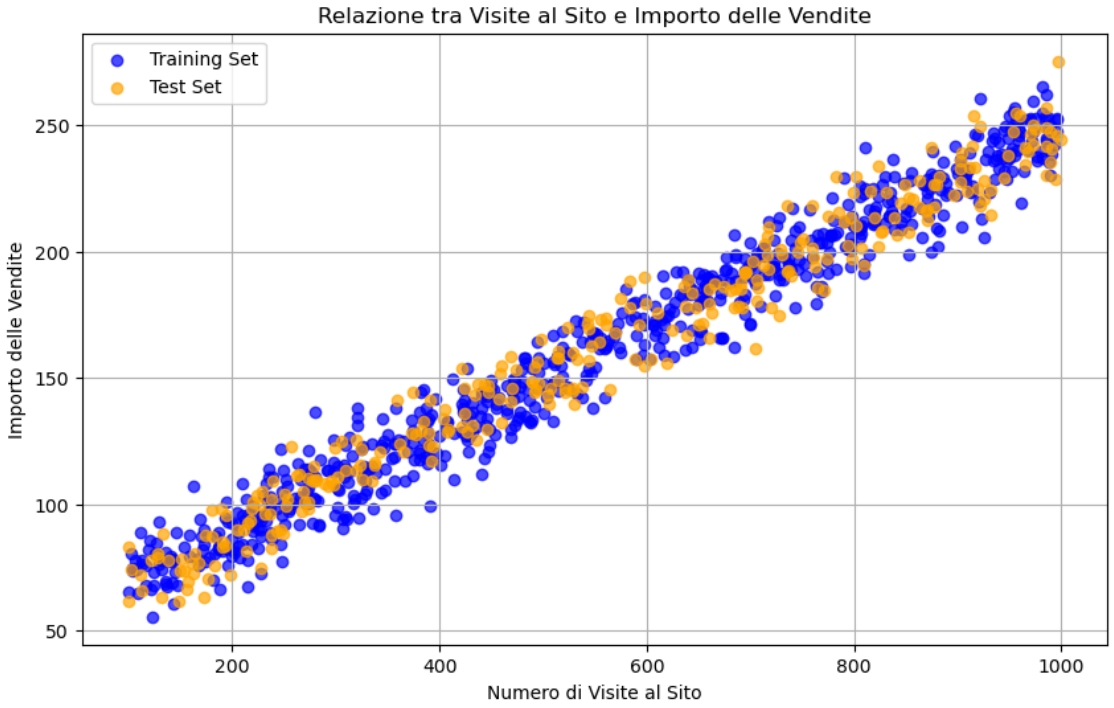
[3]:
```
import numpy as np
from sklearn.model_selection import train_test_split

# 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,)
```

5.2 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)
```

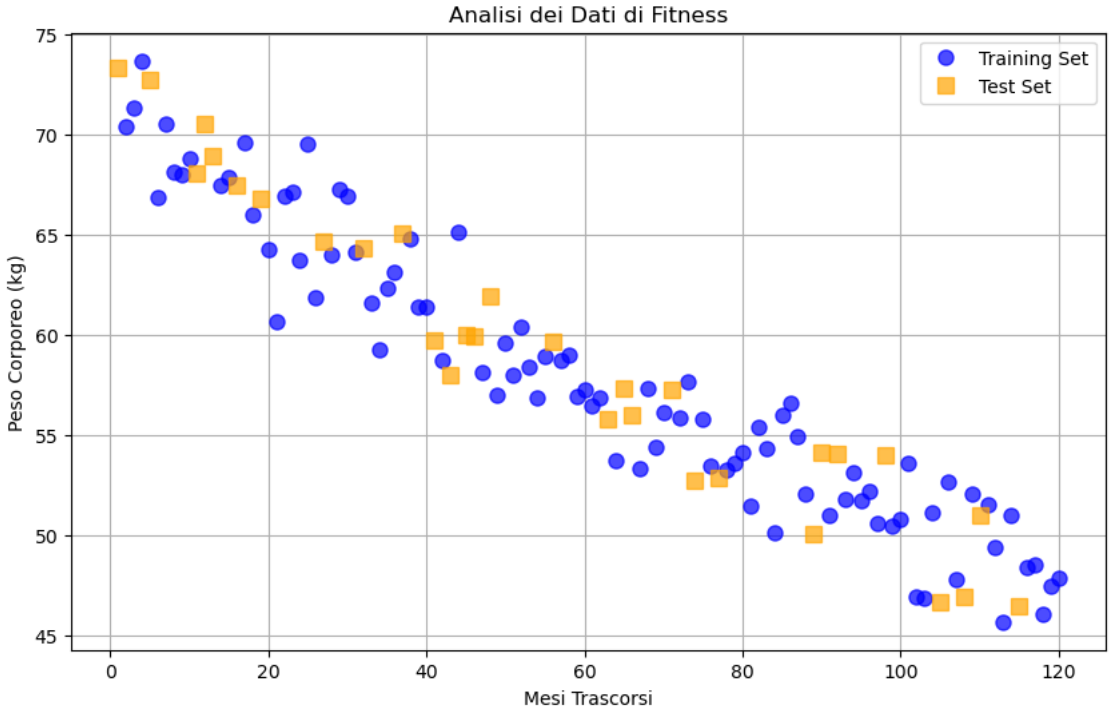



```
Dimensioni del Training Set (visite al sito e importo delle vendite): (700,)
(700,)
Dimensioni del Test Set (visite al sito e importo delle vendite): (300,) (300,)
```

5.3 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%)
```

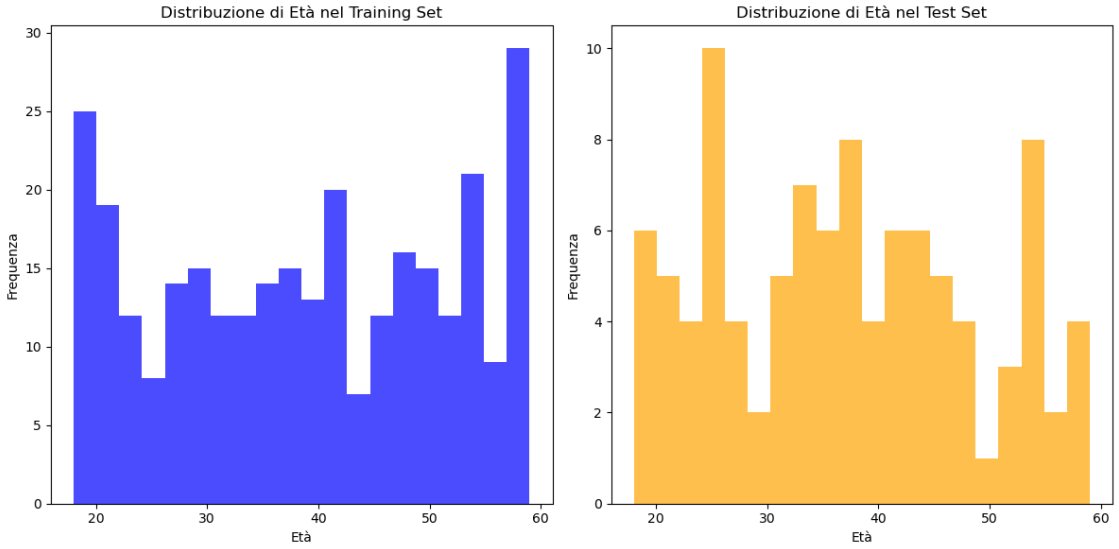
```
X_train, X_test, y_train, y_test = train_test_split(mesi_trascorsi,␣
 ↪peso_corporeo, test_size=0.25, random_state=42)

# 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,)
```

5.4 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()
```



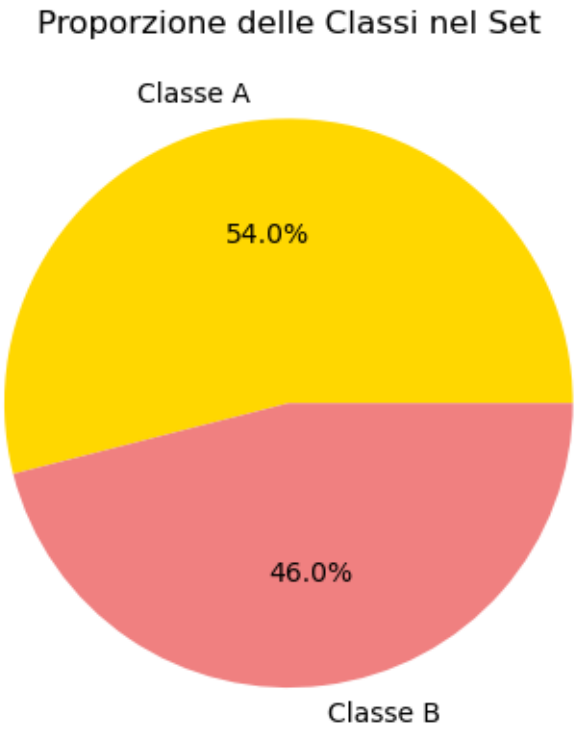
5.5 Split Stratificato dei Dati per il Training set e il Test set

[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
```

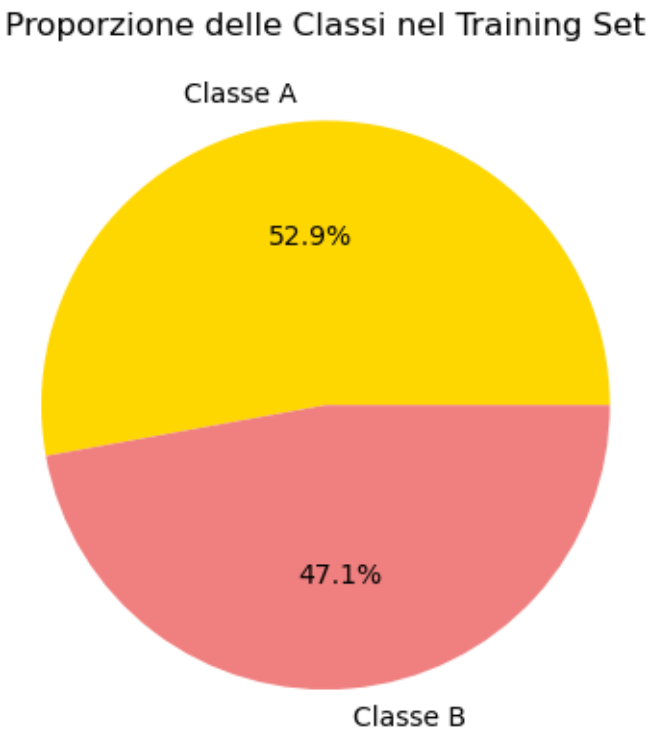
5.6 Grafico di distribuzione delle Classi nel Set

[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()
```



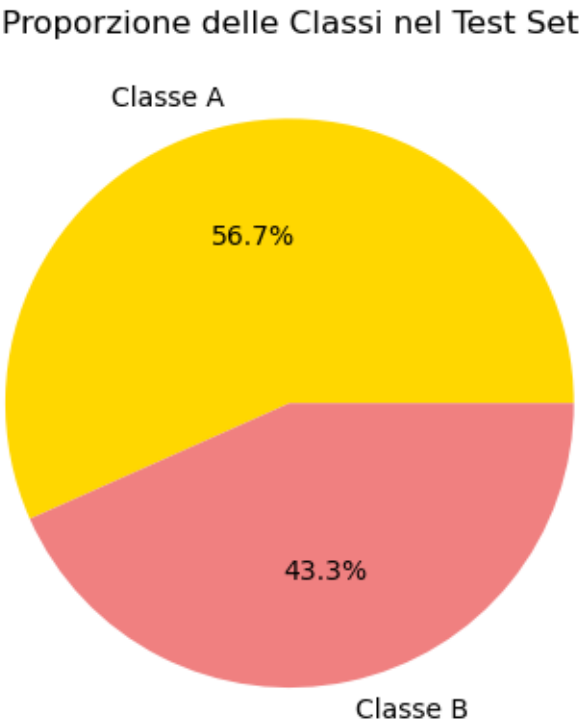
5.7 Grafico di distribuzione delle Classi nel Training Set

[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()
```



5.8 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()
```



5.9 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
```

5.10 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

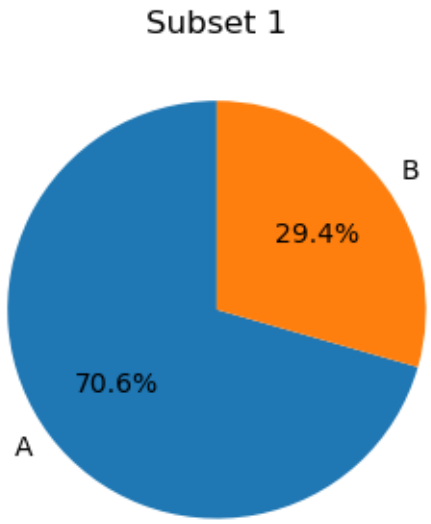
```
[100000 rows x 1 columns]
```

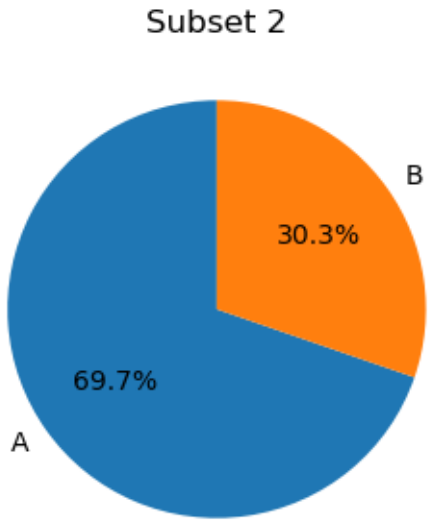
5.11 Creazione di Subset di Dimensioni Simili da un DataFrame

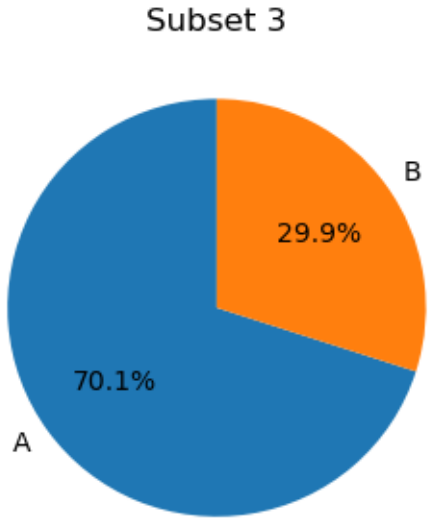
[13]:
```
# Creare tre subset di dimensioni simili
subset1 = df.sample(frac=1/3)
df = df.drop(subset1.index)

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

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

5.12 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()
```

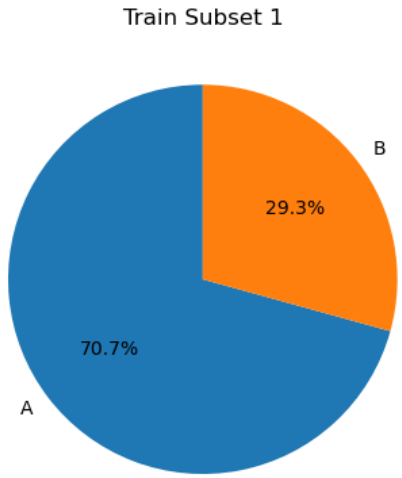

Subset 1
B
29.4%
70.6%
A

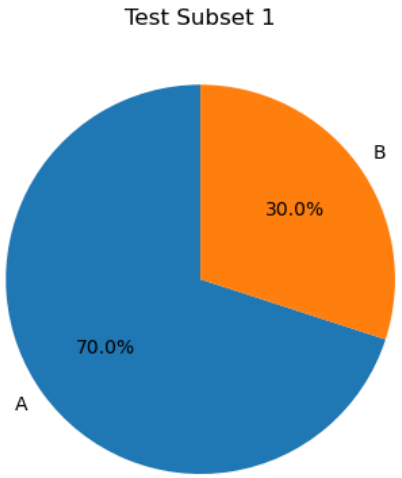

Subset 2
B
30.3%
69.7%
A

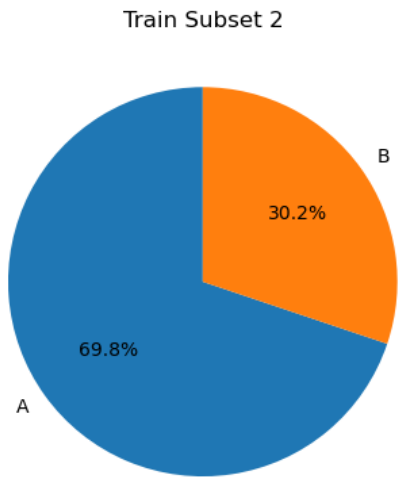

Subset 3
B
29.9%
70.1%
A

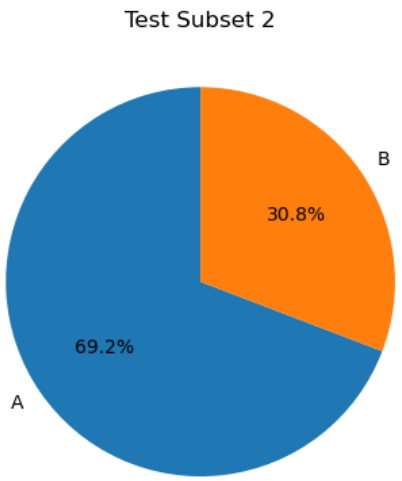
5.13 Divisione dei Subset in Training Set e Test Set con Analisi delle Distribuzioni delle Classi

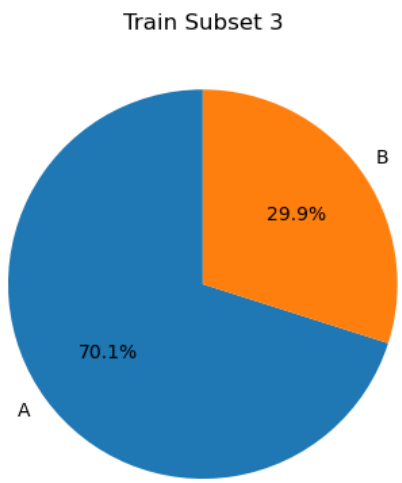
[15]:
```
# Dividere ciascun subset in training set e test set
train_subset1, test_subset1 = train_test_split(subset1, test_size=0.2,␣
↪random_state=42)
train_subset2, test_subset2 = train_test_split(subset2, test_size=0.2,␣
↪random_state=42)
train_subset3, test_subset3 = train_test_split(subset3, test_size=0.2,␣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

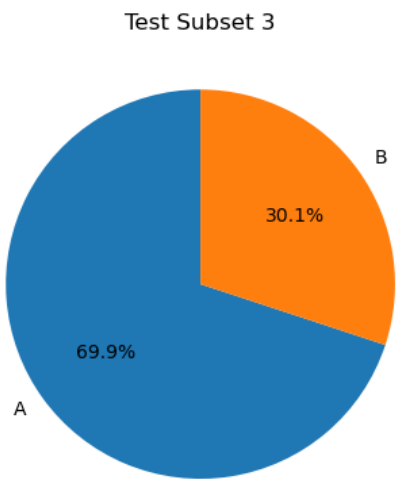

Train Subset 1
B
29.3%
70.7%
A


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

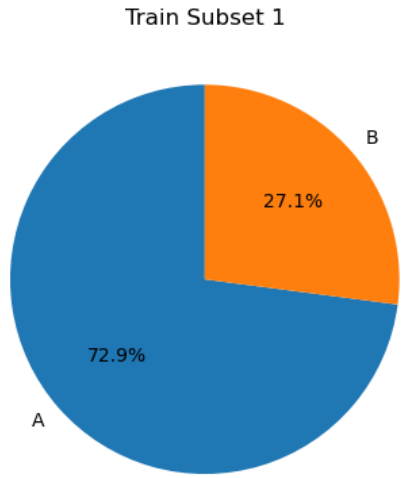

Train Subset 3
B
29.9%
70.1%
A

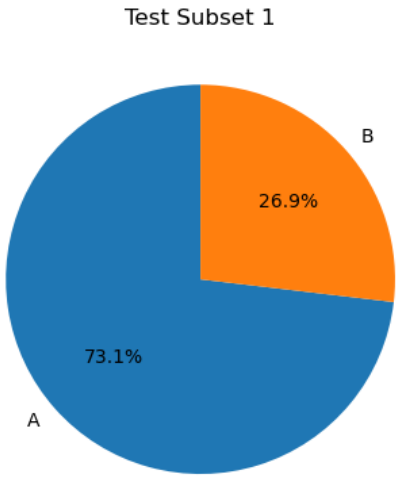

Test Subset 3
B
30.1%
69.9%
A

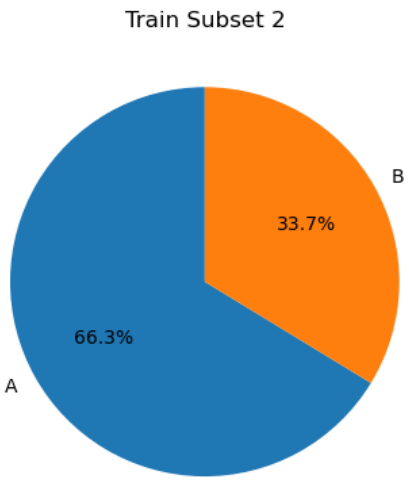
5.14 Divisione dei Subset Stratificati in Training Set e Test Set con Analisi delle Distribuzioni delle Classi

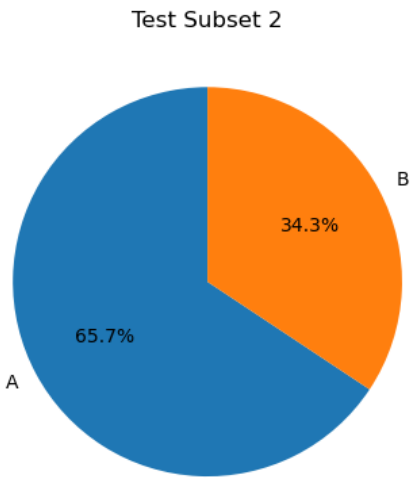
[16]:
```
np.random.seed(41)

# 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')
```

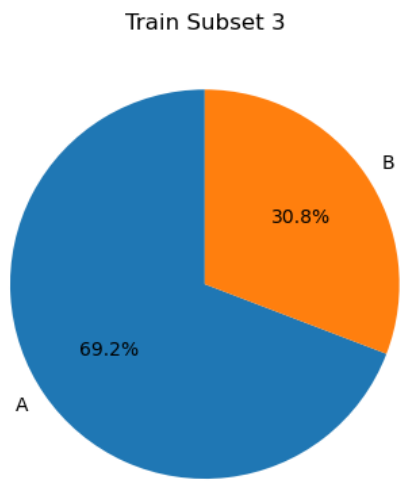
```
draw_pie(axs[1, 1], test_subset2['ColonnaAB'].value_counts(normalize=True),␣
 ↪'Test Subset 2')

# 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()
```

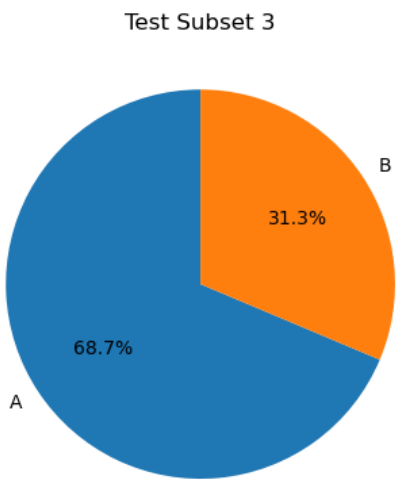

Train Subset 1
B
27.1%
72.9%
A


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

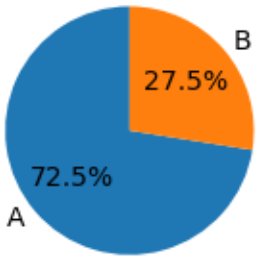

Test Subset 3
B
31.3%
68.7%
A

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

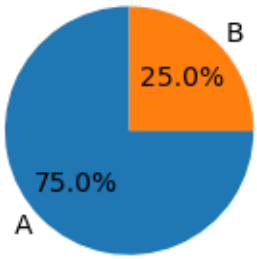
[17]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

# 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()
```

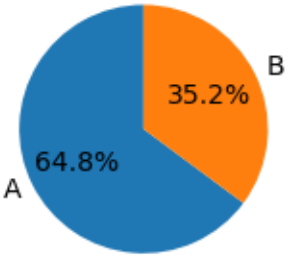
Train Subset 1



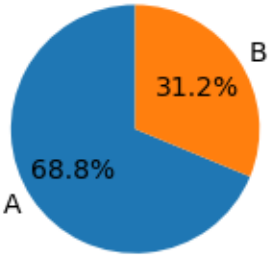
Test Subset 1



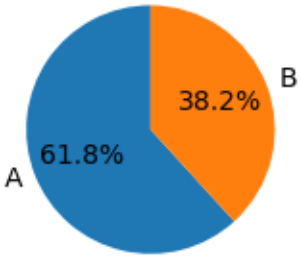
Train Subset 2



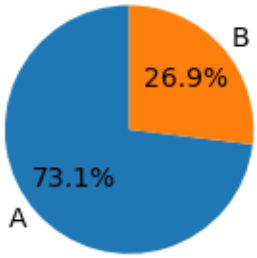
Test Subset 2



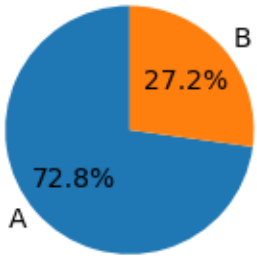
Train Subset 3



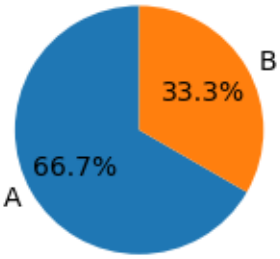
Test Subset 3



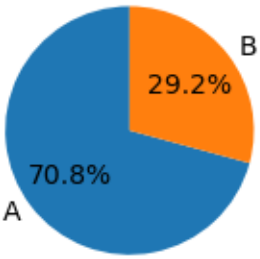
Train Subset 4



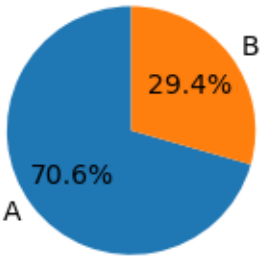
Test Subset 4



Train Subset 5



Test Subset 5



6 Extra

---

7 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.        ]
```

8 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
```

9 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]
```

10 Outliers

10.1 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
```

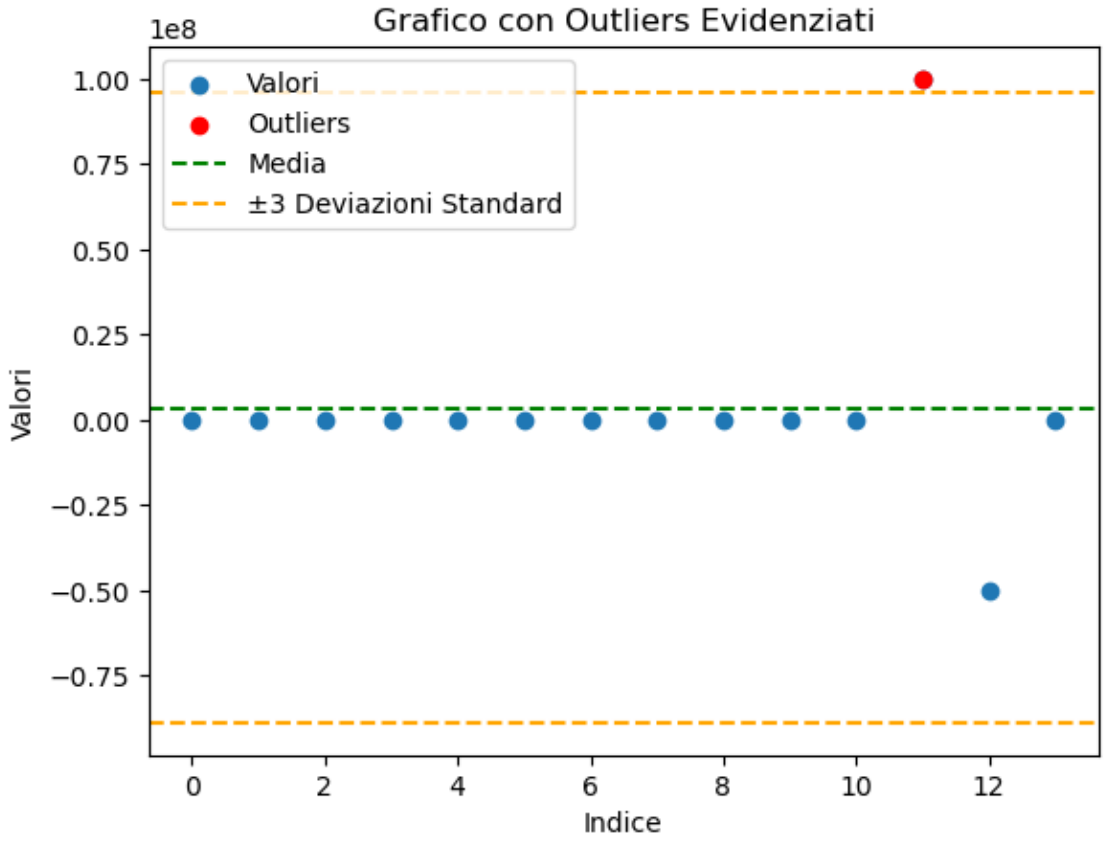
[2]:

	Valori
11	100000000

10.2 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()
```



11 Metodi per rilevare gli Outliers

---

12 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

12.1 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

12.2 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

12.3 Identificazione e rimozione degli Outlier in un DataFrame

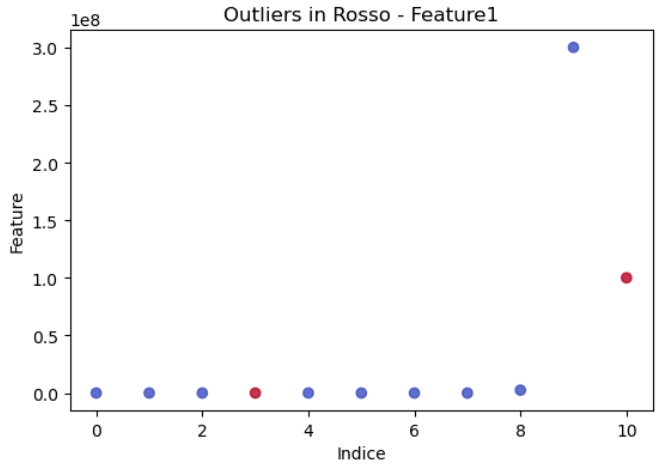
[7]:
```
# Aggiunge una colonna che indica se il record è un outlier o meno
df['Is_Outlier'] = df.index.isin(outliers.index)
# Rimuovi colonne ausiliarie
df.drop(df.filter(like='Outlier_').columns, axis=1, inplace=True)
df.drop('Num_Outliers', axis=1, inplace=True)
df
```

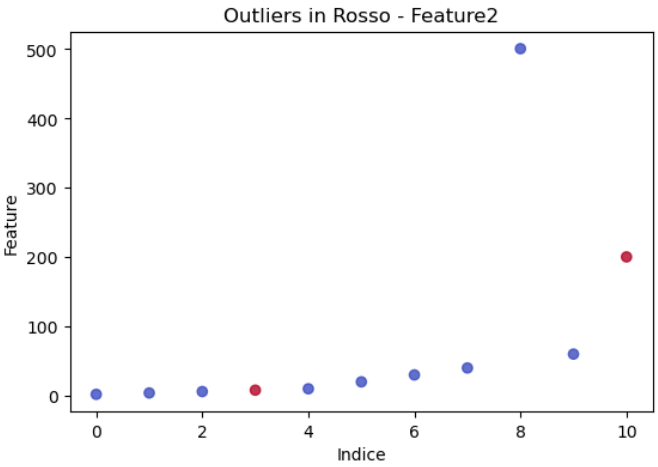
[7]:

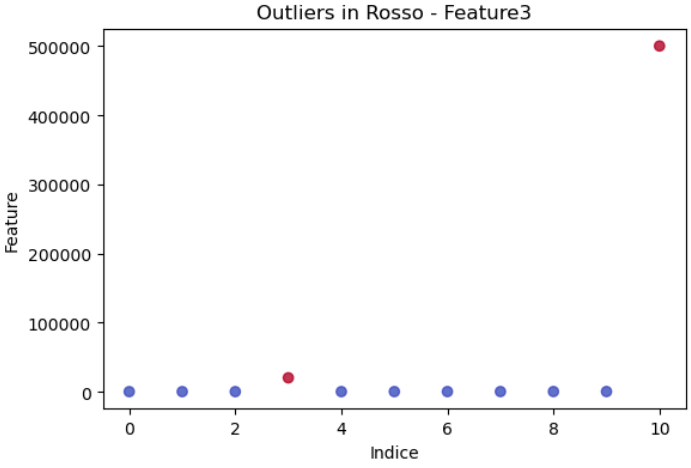
	Feature1	Feature2	Feature3	Feature4	Is_Outlier
0	1	2	5	1	False
1	200	4	10	-200000	False
2	3	6	15	3	False
3	4	8	20000	4000000000	True
4	50000	10	25	5	False
5	10	20	50	10	False
6	15	30	75	15	False
7	20	40	100	20	False
8	2500000	500	125	200	False
9	300000000	60	150	30	False
10	100000000	200	500000	10000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

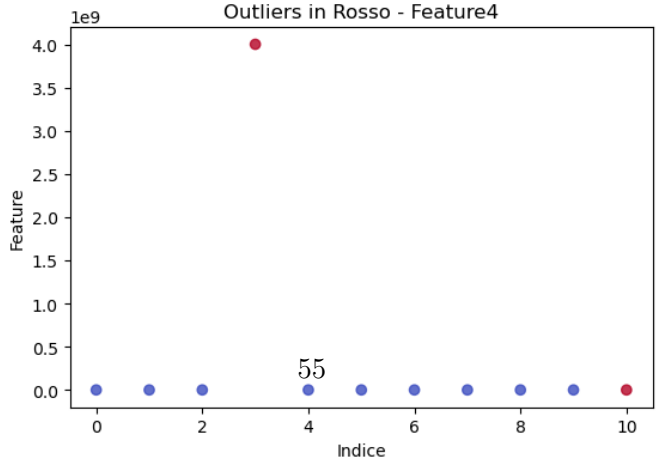
12.4 Visualizzazione Matrice dei Grafici con indicazione degli Outlier

[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

12.5 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

12.6 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
```

13 Extra

---

14 Scarto interquartile (IQR)

14.1 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
```

14.2 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
```

14.3 Calcolo dell'IQR per più colonne di un DataFrame

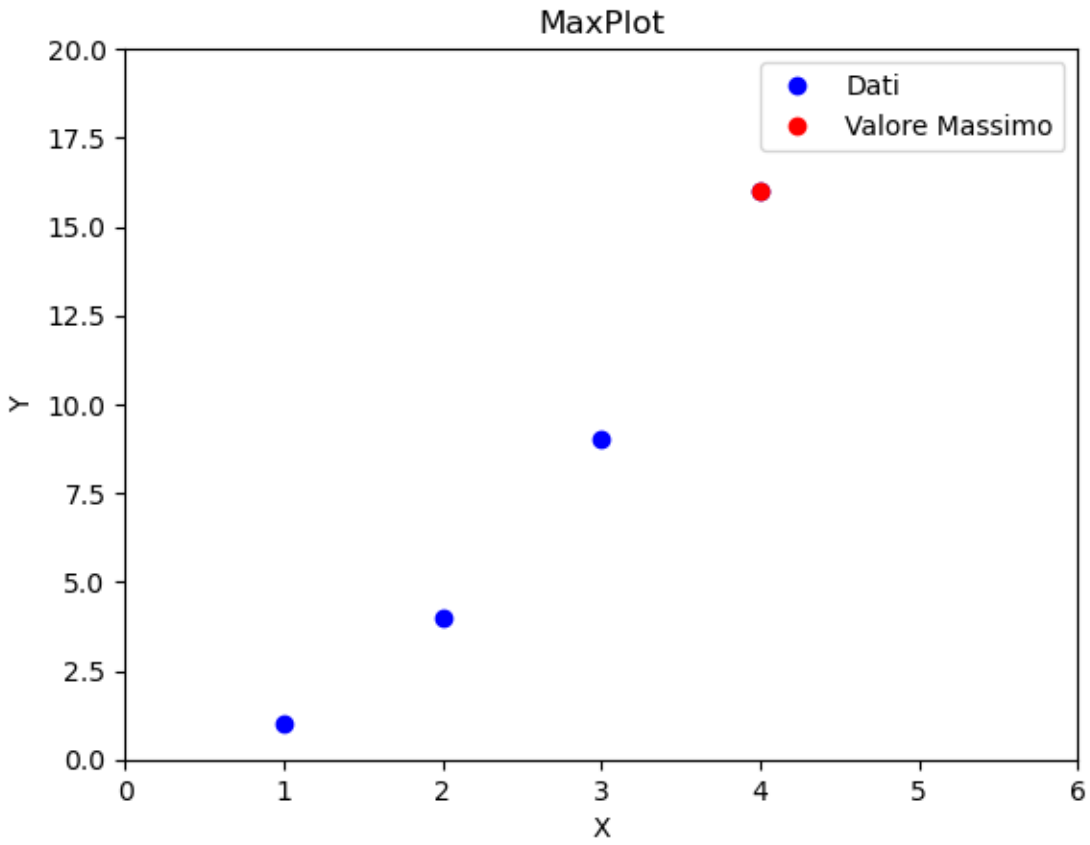
[13]:
```
# Definizione della funzione find_iqr(x) che calcola l'IQR di una serie di dati x
def find_iqr(x):
    return np.subtract(*np.percentile(x, [75, 25]))

# 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
```

15 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