ULTIMA CONSEGNA

CHATIR Ayoub

May 16, 2024

INTRODUZIONE

L'analisi dei dati è una componente fondamentale in un'ampia gamma di discipline scientifiche e commerciali, e il formato CSV (Comma-Separated Values) rappresenta uno dei modi più diffusi per la memorizzazione e lo scambio di dati tabellari. Tuttavia, i dataset possono spesso presentare lacune o valori mancanti, il che può compromettere l'accuratezza e l'affidabilità delle analisi.

In Python, la gestione dei dati mancanti è essenziale per garantire risultati significativi. La libreria Pandas offre una vasta gamma di strumenti per lavorare con dati tabellari, inclusa la gestione dei dati mancanti. Attraverso Pandas, è possibile caricare un file CSV in un DataFrame, una struttura dati tabellare potente e flessibile.

Quando si lavora con dati CSV in Python, è comune incontrare valori mancanti rappresentati come NaN (Not a Number) o None. Pandas fornisce metodi per individuare, gestire e trattare questi valori mancanti. Ad esempio, il metodo 'isnull()' può essere utilizzato per identificare i valori mancanti all'interno di un DataFrame, mentre 'fillna()' consente di sostituire i valori mancanti con valori appropriati, come la media o la mediana della colonna.

Affrontare in modo efficace i dati mancanti è cruciale per ottenere risultati accurati e significativi dalle analisi dei dati. Utilizzando Pandas e altre librerie Python, gli analisti possono gestire i dati mancanti in modo efficiente, consentendo loro di concentrarsi sull'identificazione di insight significativi e di trarre conclusioni valide dalle loro analisi. Questo approccio consente di ottenere un quadro completo e affidabile dei dati, fornendo una base solida per prendere decisioni informate e guidare il progresso scientifico e commerciale.

INDICE

1 IMPORTO IL DATASET

1.1 Analisi e Visualizzazione dei Dati del Dataset Pokémon

Questo codice esegue un'analisi preliminare del dataset dei Pokémon. Inizia importando le librerie necessarie, carica il dataset da un file CSV e quindi fornisce una panoramica delle prime righe del dataset e delle informazioni generali su di esso, compresi i tipi di dati e la presenza di valori nulli.

[50]:
```
# Importazione delle Librerie Necessarie
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split

# Caricamento del Dataset dei Pokémon
df = pd.read_csv(r"C:\Users\utente\Downloads\pokemon-sporco.csv")
print(df.head())

# Esplorazione Iniziale del Dataset
print(df.info())
print(df.describe())
```

```
      #                   Name   Type 1  Type 2  Total  HP  Attack  Defense  \
0     1              Bulbasaur    Grass  Poison    318  45      49       49
1     2                Ivysaur    Grass  Poison    405  60      62       63
2     3               Venusaur    Grass  Poison    525  80      82       83
3     3  VenusaurMega Venusaur    Grass  Poison    625  80     100      123
4     4             Charmander     Fire     NaN    309  39      52       43
..  ...                    ...      ...     ...    ...  ..     ...      ...
795  719               Diancie     Rock   Fairy    600  50     100      150
796  719    DiancieMega Diancie     Rock   Fairy    700  50     160      110
797  720    HoopaHoopa Confined  Psychic   Ghost    600  80     110       60
798  720     HoopaHoopa Unbound  Psychic    Dark    680  80     160       60
799  721              Volcanion     Fire   Water    600  80     110      120

     Sp. Atk  Sp. Def  Speed  Generation  Legendary
0         65       65     45           1      False
```

```
1          80       80     60           1      False
2         100      100     80           1      False
3         122      120     80           1      False
4          60       50     65           1      False
..        ...      ...    ...         ...        ...
795       100      150     50           6       True
796       160      110    110           6       True
797       150      130     70           6       True
798       170      130     80           6       True
799       130       90     70           6       True

[800 rows x 13 columns]
```

1.2 Gestione e Conteggio dei Dati Mancanti nel Dataset Pokémon

In questo script, esamineremo il dataset Pokémon per identificare e analizzare le righe che contengono dati mancanti. Questo passaggio è cruciale per garantire la qualità e l'integrità dei dati prima di procedere con ulteriori analisi o modellazioni.

[51]:
```
# Identificazione delle Righe con Dati Mancanti
righedatimancanti = df[df.isnull().any(axis=1)]  # any(axis=1) è True se almeno un valore nella riga è mancante (None o NaN)

# Calcolo del Numero Totale di Righe con Dati Mancanti
totaledatimancanti = righedatimancanti.shape[0]  # fornisce il numero di righe nel DataFrame

# Stampa delle Righe con Dati Mancanti
print("Le righe con i dati mancanti sono: ")
print(righedatimancanti)

# Stampa del Totale delle Righe con Dati Mancanti
print(f"Il totale dei dati mancanti è: {totaledatimancanti}")
```

```
Le righe con i dati mancanti sono:
       #        Name  Type 1 Type 2  Total   HP  Attack  Defense  Sp. Atk  \
4      4  Charmander    Fire    NaN    309   39      52       43       60
5      5  Charmeleon    Fire    NaN    405   58      64       58       80
9      7    Squirtle   Water    NaN    314   44      48       65       50
10     8   Wartortle   Water    NaN    405   59      63       80       65
11     9   Blastoise   Water    NaN    530   79      83      100       85
..   ...         ...     ...    ...    ...  ...     ...      ...      ...
775  705     Sliggoo  Dragon    NaN    452   68      75       53       83
776  706      Goodra  Dragon    NaN    600   90     100       70      110
788  712    Bergmite     Ice    NaN    304   55      69       85       32
789  713     Avalugg     Ice    NaN    514   95     117      184       44
792  716     Xerneas   Fairy    NaN    680  126     131       95      131
```

```
     Sp. Def  Speed  Generation  Legendary
4          50     65           1      False
5          65     80           1      False
9          64     43           1      False
10         80     58           1      False
11        105     78           1      False
..        ...    ...         ...        ...
775       113     60           6      False
776       150     80           6      False
788        35     28           6      False
789        46     28           6      False
792        98     99           6       True

[386 rows x 13 columns]
Il totale dei dati mancanti è: 386
```

1.3 Analisi dei Dati Mancanti nel Dataset

Questo script Python è progettato per analizzare e identificare i dati mancanti all'interno di un dataset di PokémonUtilizzando la libreria Pandas, vengono caricati i dati da un file CSV. Successivamente, vengono calcolati il totale delle righe con dati mancanti e le colonne che contengono almeno un valore mancante. Queste informazioni sono preziose per comprendere l'integrità dei dati e guidare il processo di pulizia e preparazione dei dati per analisi e modellazione successiva. La stampa dei risultati fornisce una panoramica immediata sui punti critici del dataset, facilitando la fase di esplorazione e preparazione dei dati.

[52]:
```
# totaledatimancanti = righedatimancanti.shape[0]
# Calcoliamo il totale delle righe che contengono dati mancanti nel DataFrame righedatimancanti.
# La proprietà .shape restituisce una tupla dove il primo elemento è il numero di righe e il secondo è il numero di colonne.
# Usiamo l'indice 0 per accedere al numero di righe.
totaledatimancanti = righedatimancanti.shape[0]

# colonnedatimancanti = df.isnull().any(axis=0)
# Identifichiamo le colonne che contengono dati mancanti nel DataFrame df.
# Il metodo .isnull() restituisce un DataFrame di valori booleani, True indica valori mancanti e False indica valori non mancanti.
# Il metodo .any(axis=0) restituisce True se almeno un valore nella colonna è mancante (None o NaN).
# L'asse 0 specifica che stiamo esaminando le colonne.
colonnedatimancanti = df.isnull().any(axis=0)

# Stampa delle colonne con dati mancanti
print("Colonne con dati mancanti:")
print(colonnedatimancanti)
```

```
# Stampa del totale dei dati mancanti
print(f"Totale dei dati mancanti: {totaledatimancanti}")
```

```
Colonne con dati mancanti:
#             False
Name          False
Type 1        False
Type 2         True
Total         False
HP            False
Attack        False
Defense       False
Sp. Atk       False
Sp. Def       False
Speed         False
Generation    False
Legendary     False
dtype: bool
Totale dei dati mancanti: 386
```

1.4 Analisi dei Valori Mancanti nel DataFrame

Questo codice esegue un'analisi dei valori mancanti all'interno di un DataFrame, con particolare attenzione alle colonne. Utilizzando la funzione df.isnull(), viene generato un DataFrame di valori booleani che indica la presenza di valori mancanti in ciascuna cella. Successivamente, viene calcolato il numero totale di valori mancanti per ciascuna colonna utilizzando df.isnull().sum(). Queste operazioni forniscono una panoramica dei dati mancanti nel DataFrame, che è fondamentale per la pulizia e la preparazione dei dati per l'analisi successiva.

[53]:
```
# La funzione df.isnull() restituisce un DataFrame in cui ogni cella contiene␣
↪True se il valore è mancante (NaN), altrimenti False.
# Questo ci fornisce una rappresentazione dei valori mancanti nel DataFrame df.

# Stampa dei valori mancanti per ciascuna colonna
print("I valori mancanti per ciascuna colonna sono:")
print(datimancanti)

# Gestione dei valori mancanti
datimancanti = df.isnull().sum()
# La funzione df.isnull().sum() restituisce il numero totale di valori mancanti␣
↪per ciascuna colonna del DataFrame df.
# Somma i valori True lungo l'asse 0, che rappresenta le colonne.

# Stampa dei valori mancanti per ciascuna colonna
print("I valori mancanti per ciascuna colonna sono:")
print(datimancanti)
```

```
I valori mancanti per ciascuna colonna sono:
         #   Name  Type 1  Type 2  Total     HP  Attack  Defense  Sp. Atk  \
0    False  False   False   False  False  False   False    False    False
1    False  False   False   False  False  False   False    False    False
2    False  False   False   False  False  False   False    False    False
3    False  False   False   False  False  False   False    False    False
4    False  False   False    True  False  False   False    False    False
..     ...    ...     ...     ...    ...    ...     ...      ...      ...
795  False  False   False   False  False  False   False    False    False
796  False  False   False   False  False  False   False    False    False
797  False  False   False   False  False  False   False    False    False
798  False  False   False   False  False  False   False    False    False
799  False  False   False   False  False  False   False    False    False

     Sp. Def  Speed  Generation  Legendary
0      False  False       False      False
1      False  False       False      False
2      False  False       False      False
3      False  False       False      False
4      False  False       False      False
..       ...    ...         ...        ...
795    False  False       False      False
796    False  False       False      False
797    False  False       False      False
798    False  False       False      False
799    False  False       False      False

[800 rows x 13 columns]
I valori mancanti per ciascuna colonna sono:
#               0
Name            0
Type 1          0
Type 2        386
Total           0
HP              0
Attack          0
Defense         0
Sp. Atk         0
Sp. Def         0
Speed           0
Generation      0
Legendary       0
dtype: int64
```

1.5 Visualizzazione dei Valori Mancanti tramite Heatmap

Questo blocco di codice utilizza Seaborn e Matplotlib per visualizzare i valori mancanti nel DataFrame df sotto forma di heatmap. La heatmap fornisce una rappresentazione visiva della

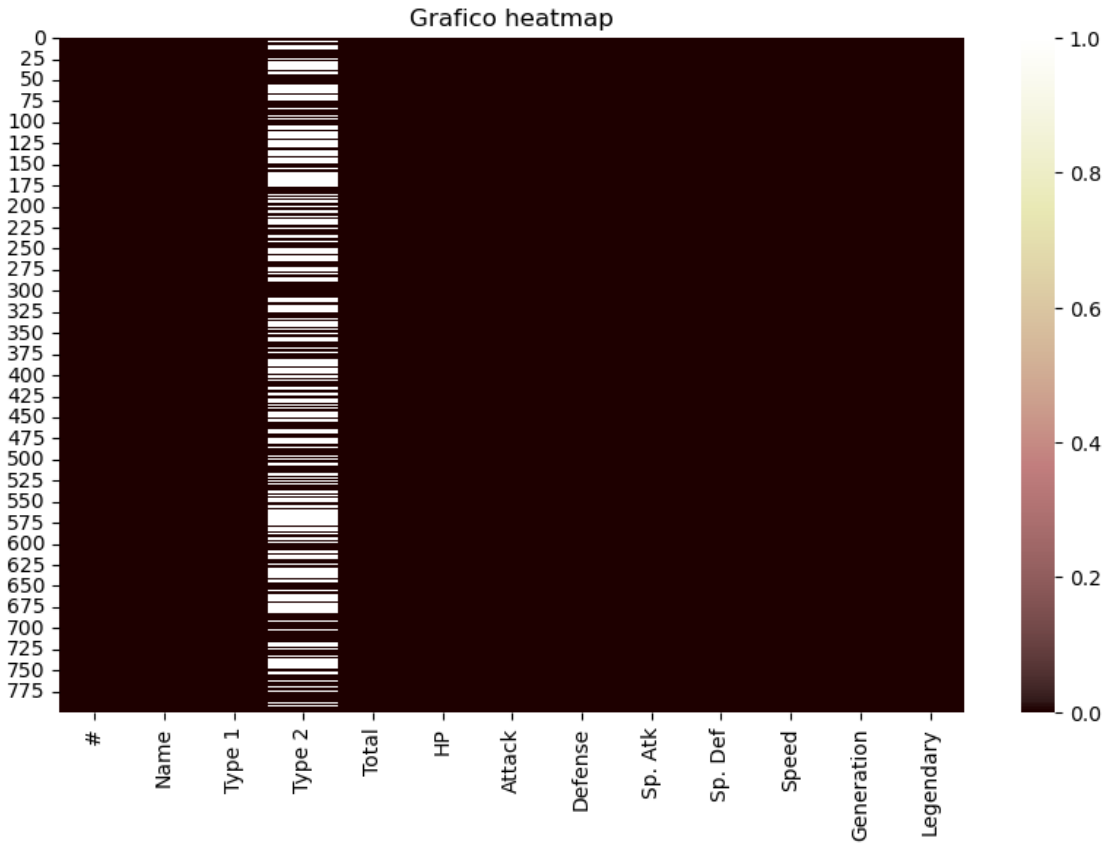
presenza dei valori mancanti in ciascuna cella del DataFrame, dove i valori mancanti sono colorati in rosa. La barra dei colori permette di interpretare facilmente la quantità di valori mancanti. Questa visualizzazione è utile per identificare rapidamente le aree in cui i dati sono mancanti nel dataset.

[54]:
```
# Creazione di una nuova figura con dimensioni 10x6 pollici utilizzando plt.
 ↪figure(figsize=(10, 6))
plt.figure(figsize=(10, 6))

# Utilizzo di seaborn per creare un heatmap che rappresenta la presenza dei␣
 ↪valori mancanti nel DataFrame df.
# Il parametro cbar=True indica che si vuole visualizzare la barra dei colori␣
 ↪per la legenda.
# Il parametro cmap="pink" imposta il colore della heatmap su rosa.
sns.heatmap(df.isnull(), cbar=True, cmap="pink")

# Impostazione del titolo del grafico su "Grafico heatmap" utilizzando plt.
 ↪title("Grafico heatmap")
plt.title("Grafico heatmap")

# Visualizzazione del grafico
plt.show()
```



1.6 Pre-elaborazione dei Dati: Gestione dei Valori Mancanti

Questo blocco di codice crea un nuovo DataFrame (df1) a partire da un DataFrame vuoto. Le colonne numeriche vengono selezionate e i valori mancanti vengono sostituiti con la media dei valori presenti nella stessa colonna. Le colonne categoriche vengono selezionate e i valori mancanti vengono sostituiti con la stringa 'None', in quanto rappresentano attributi che potrebbero non essere presenti per tutti gli elementi, come ad esempio il secondo tipo di Pokémon. Infine, vengono stampati sia il DataFrame originale con i valori mancanti, sia il nuovo DataFrame con i valori mancanti sostituiti.

[55]:
```
# Creazione di un nuovo DataFrame vuoto utilizzando pd.DataFrame()
df1 = pd.DataFrame()

# Selezione delle colonne numeriche e sostituzione dei valori NaN con la media
# Utilizzando select_dtypes(include=['number']), vengono selezionate solo le␣
 ↪colonne numeriche del DataFrame originale df.
colonnenumeriche = df.select_dtypes(include=['number'])
# Il metodo fillna(colonnenumeriche.mean()) riempie i valori NaN nelle colonne␣
 ↪numeriche con la media dei valori presenti nella stessa colonna.
df1[colonnenumeriche.columns] = colonnenumeriche.fillna(colonnenumeriche.mean())

# Selezione delle colonne categoriche e sostituzione dei valori NaN con 'None'
# Utilizzando select_dtypes(include=['object']), vengono selezionate solo le␣
 ↪colonne categoriche (oggetti) del DataFrame originale df.
colonnecategoriche = df.select_dtypes(include=['object'])
# Il metodo fillna('None') riempie i valori NaN nelle colonne categoriche con la␣
 ↪stringa 'None'.
# Questo è utile nel caso di colonne che rappresentano attributi che potrebbero␣
 ↪non essere presenti per tutti gli elementi, come il secondo tipo di Pokémon.
df1[colonnecategoriche.columns] = colonnecategoriche.fillna('None')

# Stampa dei DataFrame originale e modificato
print(f"Questo è il primo DataFrame con i valori mancanti: \n{df}\nQuesto invece␣
 ↪è il secondo DataFrame con i missing values sostituiti:\n{df1}\n")
```

```
Questo è il primo con i valori mancanti:
       #                    Name   Type 1  Type 2  Total  HP  Attack  Defense  \
0      1               Bulbasaur    Grass  Poison    318  45      49       49
1      2                 Ivysaur    Grass  Poison    405  60      62       63
2      3                Venusaur    Grass  Poison    525  80      82       83
3      3  VenusaurMega Venusaur    Grass  Poison    625  80     100      123
4      4              Charmander     Fire     NaN    309  39      52       43
..   ...                     ...      ...     ...    ...  ..     ...      ...
795  719                 Diancie     Rock   Fairy    600  50     100      150
796  719    DiancieMega Diancie     Rock   Fairy    700  50     160      110
797  720    HoopaHoopa Confined  Psychic   Ghost    600  80     110       60
```

```
798  720     HoopaHoopa Unbound  Psychic    Dark    680  80     160       60
799  721              Volcanion     Fire   Water    600  80     110      120

     Sp. Atk  Sp. Def  Speed  Generation  Legendary
0         65       65     45           1      False
1         80       80     60           1      False
2        100      100     80           1      False
3        122      120     80           1      False
4         60       50     65           1      False
..       ...      ...    ...         ...        ...
795      100      150     50           6       True
796      160      110    110           6       True
797      150      130     70           6       True
798      170      130     80           6       True
799      130       90     70           6       True

[800 rows x 13 columns]
Questo invece è il secondo con i missing values sostituiti:
       #  Total  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed  Generation  \
0      1    318  45      49       49       65       65     45           1
1      2    405  60      62       63       80       80     60           1
2      3    525  80      82       83      100      100     80           1
3      3    625  80     100      123      122      120     80           1
4      4    309  39      52       43       60       50     65           1
..   ...    ...  ..     ...      ...      ...      ...    ...         ...
795  719    600  50     100      150      100      150     50           6
796  719    700  50     160      110      160      110    110           6
797  720    600  80     110       60      150      130     70           6
798  720    680  80     160       60      170      130     80           6
799  721    600  80     110      120      130       90     70           6

                     Name   Type 1  Type 2
0               Bulbasaur    Grass  Poison
1                 Ivysaur    Grass  Poison
2                Venusaur    Grass  Poison
3    VenusaurMega Venusaur    Grass  Poison
4              Charmander     Fire    None
..                    ...      ...     ...
795               Diancie     Rock   Fairy
796    DiancieMega Diancie     Rock   Fairy
797    HoopaHoopa Confined  Psychic   Ghost
798     HoopaHoopa Unbound  Psychic    Dark
799              Volcanion     Fire   Water

[800 rows x 12 columns]
```

1.7 DataFrame con Valori Mancanti Gestiti

[56]:
```
df1
```

[56]:
```
       #  Total  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed  Generation  \
0      1    318  45      49       49       65       65     45           1
1      2    405  60      62       63       80       80     60           1
2      3    525  80      82       83      100      100     80           1
3      3    625  80     100      123      122      120     80           1
4      4    309  39      52       43       60       50     65           1
..   ...    ...  ..     ...      ...      ...      ...    ...         ...
795  719    600  50     100      150      100      150     50           6
796  719    700  50     160      110      160      110    110           6
797  720    600  80     110       60      150      130     70           6
798  720    680  80     160       60      170      130     80           6
799  721    600  80     110      120      130       90     70           6

                      Name   Type 1  Type 2
0                Bulbasaur    Grass  Poison
1                  Ivysaur    Grass  Poison
2                 Venusaur    Grass  Poison
3    VenusaurMega Venusaur    Grass  Poison
4               Charmander     Fire    None
..                     ...      ...     ...
795               Diancie     Rock   Fairy
796    DiancieMega Diancie     Rock   Fairy
797    HoopaHoopa Confined  Psychic   Ghost
798     HoopaHoopa Unbound  Psychic    Dark
799              Volcanion     Fire   Water

[800 rows x 12 columns]
```

1.8 Individuazione degli Outliers per le Caratteristiche del Pokémon

Questo blocco di codice esegue un'analisi per individuare gli outliers nelle caratteristiche dei Pokémon come 'Total', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def' e 'Speed'. Per ciascuna caratteristica, vengono calcolate la media e la deviazione standard, e successivamente vengono identificati gli outliers utilizzando un criterio basato sulla media e sulla deviazione standard (considerando gli outliers quelli che si discostano di più di 3 deviazioni standard dalla media). Infine, tutti gli outliers identificati vengono concatenati in un unico DataFrame outliers.

[57]:
```
# Calcolo della media e della deviazione standard per la caratteristica 'Total'
mean_value1 = df1['Total'].mean()
std_dev = df1['Total'].std()
# Identificazione degli outliers per 'Total' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers1 = df1[(df1['Total'] > mean_value + 3 * std_dev) | (df1['Total'] <␣
 ↪mean_value - 3 * std_dev)]
```

```
# Calcolo della media e della deviazione standard per la caratteristica 'Attack'
mean_value2 = df1['Attack'].mean()
std_dev = df1['Attack'].std()
# Identificazione degli outliers per 'Attack' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers2 = df1[(df1['Attack'] > mean_value + 3 * std_dev) | (df1['Attack'] <␣
 ↪mean_value - 3 * std_dev)]

# Calcolo della media e della deviazione standard per la caratteristica 'Defense'
mean_value3 = df1['Defense'].mean()
std_dev = df1['Defense'].std()
# Identificazione degli outliers per 'Defense' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers3 = df1[(df1['Defense'] > mean_value + 3 * std_dev) | (df1['Defense'] <␣
 ↪mean_value - 3 * std_dev)]

# Calcolo della media e della deviazione standard per la caratteristica 'Sp. Atk'
mean_value4 = df1['Sp. Atk'].mean()
std_dev = df1['Sp. Atk'].std()
# Identificazione degli outliers per 'Sp. Atk' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers4 = df1[(df1['Sp. Atk'] > mean_value + 3 * std_dev) | (df1['Sp. Atk'] <␣
 ↪mean_value - 3 * std_dev)]

# Calcolo della media e della deviazione standard per la caratteristica 'Sp. Def'
mean_value5 = df1['Sp. Def'].mean()
std_dev = df1['Sp. Def'].std()
# Identificazione degli outliers per 'Sp. Def' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers5 = df1[(df1['Sp. Def'] > mean_value + 3 * std_dev) | (df1['Sp. Def'] <␣
 ↪mean_value - 3 * std_dev)]

# Calcolo della media e della deviazione standard per la caratteristica 'Speed'
mean_value6 = df1['Speed'].mean()
std_dev = df1['Speed'].std()
# Identificazione degli outliers per 'Speed' utilizzando un criterio basato␣
 ↪sulla media e sulla deviazione standard
outliers6 = df1[(df1['Speed'] > mean_value + 3 * std_dev) | (df1['Speed'] <␣
 ↪mean_value - 3 * std_dev)]

# Concatenazione di tutti gli outliers identificati per le varie caratteristiche
outliers = pd.concat([outliers1, outliers2, outliers3, outliers4, outliers5,␣
 ↪outliers6])
```

[57]:
```
       #  Total  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed  Generation  \
315  291    456  61      90       45       50       50    160           3
431  386    600  50      95       90       95       90    180           3

                   Name   Type 1  Type 2
315            Ninjask      Bug  Flying
431  DeoxysSpeed Forme  Psychic    None
```

1.9 Grafico a Dispersione delle Caratteristiche dei Pokémon con Evidenziazione degli Outliers

Questo blocco di codice crea un grafico a dispersione per ciascuna caratteristica del Pokémon, evidenziando gli outliers in rosso. Viene anche aggiunta una linea per la media e ± 3 deviazioni standard per la prima caratteristica 'Total'. Questo grafico fornisce una rappresentazione visiva delle distribuzioni delle caratteristiche del Pokémon e dei loro outliers, che sono stati individuati nel blocco di codice precedente.

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

# Crea un grafico a dispersione per il primo set di dati
plt.scatter(df.index, mean_value1)

# Evidenzia gli outliers nel grafico con un colore diverso
plt.scatter(outliers.index, outliers1, color='red')

# Crea un grafico a dispersione per il secondo set di dati
plt.scatter(df.index, mean_value2)

# Evidenzia gli outliers nel grafico con un colore diverso
plt.scatter(outliers.index, outliers2, color='red')

# Crea un grafico a dispersione per il terzo set di dati
plt.scatter(df.index, mean_value3)

# Evidenzia gli outliers nel grafico con un colore diverso
plt.scatter(outliers.index, outliers3, color='red')

# Crea un grafico a dispersione per il quarto set di dati
plt.scatter(df.index, mean_value4)

# Evidenzia gli outliers nel grafico con un colore diverso
plt.scatter(outliers.index, outliers4, color='red')

# Crea un grafico a dispersione per il quinto set di dati
plt.scatter(df.index, mean_value5)

# Evidenzia gli outliers nel grafico con un colore diverso
```

```
plt.scatter(outliers.index, outliers5, color='red')

# Crea un grafico a dispersione per il sesto set di dati
plt.scatter(df.index, mean_value6)

# Evidenzia gli outliers nel grafico con un colore diverso
plt.scatter(outliers.index, outliers6, color='red')

# Aggiungi la media e la deviazione standard al grafico
plt.axhline(y=mean_value1, color='green', linestyle='--', label='Media')
plt.axhline(y=mean_value1 + 3 * std_dev, color='orange', linestyle='--',␣
↪label='±3 Deviazioni Standard')
plt.axhline(y=mean_value1 - 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()
```

1.10 Suddivisione dei Dati in Set di Addestramento e Test

Questo blocco di codice importa la funzione train_test_split da sklearn.model_selection per suddividere i dati in set di addestramento e set di test. Vengono selezionate le caratteristiche rilevanti come variabili indipendenti per l'addestramento del modello. Successivamente, i dati vengono divisi in set di addestramento e set di test utilizzando una proporzione del 70% per il set di addestramento e del 30% per il set di test. Infine, le dimensioni dei set di addestramento e test vengono stampate a fini di verifica.

[ ]:
```
from sklearn.model_selection import train_test_split

# Selezione delle caratteristiche da utilizzare come variabili indipendenti
Total = df1['Total']
HP = df1['HP']
Attack = df1['Attack']
Defense = df1['Defense']
SpecialAtk = df1['Sp. Atk']
SpecialDef = df1['Sp. Def']
Speed = df1['Speed']
Generation = df1['Generation']
Name = df1['Name']
Type_1 = df1['Type 1']
Type_2 = df1['Type 2']
```

```
# Splitting dei dati in set di addestramento e set di test
a_train, a_test, s_train, s_test, d_train, d_test, f_train, f_test, g_train,␣
 ↪g_test, h_train, h_test, j_train, j_test, k_train, k_test, l_train, l_test,␣
 ↪z_train, z_test, x_train, x_test = train_test_split(Total, HP, Attack,␣
 ↪Defense, SpecialAtk, SpecialDef, Speed, Generation, Name, Type_1, Type_2,␣
 ↪test_size=0.3, random_state=42)

# Stampa delle dimensioni del set di addestramento e del set di test
print("Dimensioni del Training Set :", a_train.shape, s_train.shape, d_train.
 ↪shape, f_train.shape, g_train.shape, h_train.shape, j_train.shape, k_train.
 ↪shape, l_train.shape, z_train.shape, x_train.shape)
print("Dimensioni del Test Set :", a_test.shape, s_test.shape, d_test.shape,␣
 ↪f_test.shape, g_test.shape, h_test.shape, j_test.shape, k_test.shape, l_test.
 ↪shape, z_test.shape, x_test.shape)
```

```
Dimensioni del Training Set : (560,) (560,) (560,) (560,) (560,) (560,) (560,)
(560,) (560,) (560,) (560,)
Dimensioni del Test Set : (240,) (240,) (240,) (240,) (240,) (240,) (240,)
(240,) (240,) (240,) (240,)
```

1.11 Split dei Dati e Calcolo delle Proporzioni delle Classi

Questo blocco di codice importa la funzione train_test_split da sklearn.model_selection per eseguire uno split stratificato dei dati in set di addestramento e test. Successivamente, calcola le proporzioni delle classi nel dataset completo e nei set di addestramento e test. Infine, stampa le proporzioni delle classi per valutare l'equilibrio delle classi nei diversi set di dati.

[65]:
```
from sklearn.model_selection import train_test_split
import numpy as np

# Importa la funzione per lo split dei dati in set di addestramento e test
# e la libreria numpy per le operazioni numeriche

# Seleziona le variabili indipendenti e dipendenti
X = a_train
Y = s_train
X_test = a_test
Y_test = s_test

# Calcola le proporzioni delle classi nel dataset completo
proporzione_classe_A = sum(Y == 'A') / len(Y)
proporzione_classe_B = 1 - proporzione_classe_A
# Calcola la proporzione delle altre classi
# Proporzione_classe_C, Proporzione_classe_D, etc...

# Esegui 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)
```

```python
# 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_C_train, Proporzione_classe_D_train, etc...

proporzione_classe_A_test = sum(Y_test == 'A') / len(Y_test)
proporzione_classe_B_test = 1 - proporzione_classe_A_test
# Proporzione_classe_C_test, Proporzione_classe_D_test, etc...

# Stampa delle proporzioni delle classi per il dataset completo, il training set␣
↪e il test set
print("Proporzioni Classe A nel dataset completo:", proporzione_classe_A)
print("Proporzioni Classe B nel dataset completo:", proporzione_classe_B)
# Stampa delle proporzioni delle altre classi nel dataset completo
# Proporzione_classe_C, Proporzione_classe_D, etc...
print("Proporzioni Classe A nel training set:", proporzione_classe_A_train)
print("Proporzioni Classe B nel training set:", proporzione_classe_B_train)
# Stampa delle proporzioni delle altre classi nel training set
# Proporzione_classe_C_train, Proporzione_classe_D_train, etc...
print("Proporzioni Classe A nel test set:", proporzione_classe_A_test)
print("Proporzioni Classe B nel test set:", proporzione_classe_B_test)
# Stampa delle proporzioni delle altre classi nel test set
# Proporzione_classe_C_test, Proporzione_classe_D_test, etc...
```

```
Proporzione Classe A nel data Set completo: 0.0
Proporzione Classe B nel data Setcompleto: 1.0
Proporzione Classe C nel data Setcompleto: 0.0
Proporzione Classe D nel data Setcompleto: 1.0
Proporzione Classe E nel data Setcompleto: 0.0
Proporzione Classe F nel data Setcompleto: 1.0
Proporzione Classe G nel data Setcompleto: 0.0
Proporzione Classe H nel data Setcompleto: 1.0
Proporzione Classe I nel data Setcompleto: 0.0
Proporzione Classe J nel data Setcompleto: 1.0
Proporzione Classe K nel data Setcompleto: 0.0
Proporzione Classe A nel Training Set: 0.0
Proporzione Classe B nel Training Set: 1.0
Proporzione Classe C nel Training Set: 0.0
Proporzione Classe D nel Training Set: 0.0
Proporzione Classe F nel Training Set: 1.0
Proporzione Classe G nel Training Set: 0.0
Proporzione Classe H nel Training Set: 1.0
Proporzione Classe I nel Training Set: 0.0
Proporzione Classe J nel Training Set: 1.0
Proporzione Classe K nel Training Set: 0.0
Proporzione Classe A nel Test Set: 0.0
Proporzione Classe B nel Test Set: 1.0
```

```
Proporzione Classe C nel Test Set: 0.0
Proporzione Classe D nel test Set: 1.0
Proporzione Classe E nel Test Set: 0.0
Proporzione Classe F nel Test Set: 1.0
Proporzione Classe G nel Test Set: 0.0
Proporzione Classe H nel Test Set: 1.0
Proporzione Classe I nel Test Set: 0.0
Proporzione Classe J nel Test Set: 1.0
Proporzione Classe K nel Test Set: 0.0
```