INDICE

6.9 Visualizzazione dei Training Set e Test Set in Suddivisioni Equilibrate: Analisi delle Proporzioni di Classi tra i Subset

1. LA DEVIAZIONE STANDARD

7.1 Calcolo della Deviazione Standard in Python

7.2 Visualizzazione degli Outliers in un DataFrame con Calcolo di Media e Deviazione Standard

7.3 Rilevamento degli Outliers tramite Criterio dei +3 Sigma dalla Media

7.4 Visualizzazione dei Valori con Evidenziazione degli Outliers

7.5 Rilevazione e Marcatura degli Outlier in un DataFrame Pandas con Analisi delle Features e Intervalli di Confidenza

7.6 Rilevazione e Rimozione degli Outliers: Identificazione e Eliminazione delle Righe con Caratteristiche Fuoriscala

7.7 Selezione delle righe con Numero Minimo di Features Superante la Soglia

7.8 Aggiunta della Colonna 'Is_Outlier' per Identificare gli Outlier nel DataFrame

7.9 Eliminazione Colonne Ausiliarie e Indicatori di Outlier dal DataFrame

7.10 Filtraggio dei dati: Esclusione degli Outliers da un DataFrame

7.11 Selezionare dati senza valori anomali: Filtraggio del DataFrame per escludere gli outliers

7.12 Visualizzazione degli Outlier nelle Feature tramite Matrice di Grafici

Per prima cosa è necessario installare la libreria "numpy"

```
pip install Pandoc

Requirement already satisfied: Pandoc in c:\users\utente\anaconda3\
lib\site-packages (2.3)
Requirement already satisfied: plumbum in c:\users\utente\anaconda3\
lib\site-packages (from Pandoc) (1.8.2)
Requirement already satisfied: ply in c:\users\utente\anaconda3\lib\
site-packages (from Pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\utente\anaconda3\
lib\site-packages (from plumbum->Pandoc) (305.1)
Note: you may need to restart the kernel to use updated packages.

pip install numpy

Requirement already satisfied: numpy in c:\users\utente\anaconda3\lib\
site-packages (1.24.3)
Note: you may need to restart the kernel to use updated packages.
```

L'APERTURA E LA LETTURA DI UN FILE CSV

Caricamento e Analisi Iniziale di Dati Pokémon da un File CSV

In questo codice viene mostrato come importare un file CSV da una cartella specifica del PC e successivamente come leggerlo (utilizzando la funzione "pd.read_csv"). Infine, il programma stampa anche le prime righe del DataFrame utilizzando il metodo "df.head()" e attraverso l'attributo "shape" si possono visualizzare le proprietà del file, cioè il numero di colonne (18) e il numero di righe (1017). Inoltre, quando si parla di DataFrame, i rispettivi termini di riga e colonna rappresentano le istanze e le feature.

```
import pandas as pd # per la gestione dei DataFrame
import numpy as np  # per la gestione delle diverse operazioni matematiche
import matplotlib.pyplot as plt # per la creazione di grafici

# Specificare il percorso del file CSV
percorsofilecsv=r"C:\Users\utente\Downloads\Pokemon_Sporco.csv"
# Leggere il file CSV in un DataFrame
df=pd.read_csv(percorsofilecsv) # per i file CSV si scrive "csv" nel pd.read
# Mostrare le prime righe del DataFrame
print(df.head()) # (df.head()) stampa solo le prime righe (istanze) del DataFrame
df.shape # visualizza le proprietà del DataFrame, quindi il numero totale di righe (istanze) e di colonne (Feature)
```

```
   #                   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


   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

(800, 13)
```

L'APERTURA E LA LETTURA DI UN FILE XLSX

Analisi delle Statistiche della Serie A: Stagione 09-10

In questo codice, è presentato un equivalente del precedente, ma con la differenza che viene importato un file XLSX, ovvero un file classico di Excel. Nell'ambito di questo programma, è incluso il parametro "sheet_name" nella funzione "pd.read_excel", il quale consente di leggere un foglio specifico del file Excel in questione. In questo caso, si tratta del foglio denominato "09-10".

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Specificare il percorso del file XLSX
percorsofileExcel=r"C:\Users\utente\Downloads\serieA.xlsm.xlsx"
# Leggere il file XLSX in un DataFrame
df=pd.read_excel(percorsofileExcel, sheet_name='09-10') # per i file
XLSX si scrive "excel" nel pd.read, il nome del programma
# Mostrare le prime righe del DataFrame
print(df.head())
```

```
   position                  team  Pt  Played  Won  Net  lose  Goals
made  \
0         1          Inter Inter  82      38   24   10     4
75
1         2            Roma Roma  80      38   24    8     6
68
2         3          Milan Milan  70      38   20   10     8
60
3         4  Sampdoria Sampdoria  67      38   19   10     9
49
4         5      Palermo Palermo  65      38   18   11     9
59

   Goals suffered  Difference goals
0              34                41
1              41                27
2              39                21
3              41                 8
4              47                12
```

Qui sotto vengono mostrare tutte le righe dell'ultimo DataFrame importato usando "df".

```
# Stampare tutte le righe del DataFrame
df
```

	position	team	Pt	Played	Won	Net	lose	Goals made
0	1	Inter Inter	82	38	24	10	4	75
1	2	Roma Roma	80	38	24	8	6	68
2	3	Milan Milan	70	38	20	10	8	60
3	4	Sampdoria Sampdoria	67	38	19	10	9	49
4	5	Palermo Palermo	65	38	18	11	9	59
5	6	Napoli Napoli	59	38	15	14	9	50
6	7	Juventus Juventus	55	38	16	7	15	55
7	8	Parma Parma	52	38	14	10	14	46
8	9	Genoa Genoa	51	38	14	9	15	57
9	10	Bari Bari	50	38	13	11	14	49
10	11	Fiorentina Fiorentina	47	38	13	8	17	48
11	12	Lazio Lazio	46	38	11	13	14	39
12	13	Catania Catania	45	38	10	15	13	44
13	14	Chievo Chievo	44	38	12	8	18	37
14	15	Udinese Udinese	44	38	11	11	16	54
15	16	Cagliari Cagliari	44	38	11	11	16	56
16	17	Bologna Bologna	42	38	10	12	16	42
17	18	Atalanta Atalanta	35	38	9	8	21	37
18	19	Siena Siena	31	38	7	10	21	40
19	20	Livorno Livorno	29	38	7	8	23	27

	Goals suffered	Difference goals
0	34	41
1	41	27
2	39	21
3	41	8
4	47	12
5	43	7

```
6                56                -1
7                51                -5
8                61                -4
9                49                 0
10               47                 1
11               43                -4
12               45                -1
13               42                -5
14               59                -5
15               58                -2
16               55               -13
17               53               -16
18               67               -27
19               61               -34
```

Per poter continuare con questa esercitazione, è necessario installare la libreria "os".

```
pip install os
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
ERROR: Could not find a version that satisfies the requirement os
(from versions: none)
ERROR: No matching distribution found for os
```

L'APERTURA E LA LETTURA DI UNA CARTELLA E DI UN COLLEGAMENTO FILE

Importazione di dati da file CSV in DataFrame e creazione di una lista di DataFrame con Python

In questo programma viene spiegato come aprire e leggere i diversi file all'interno di una cartella (in questo caso solo CSV) per poi poterli unire in un unico DataFrame. Inizialmente viene creata una lista di cui conterrà tutti i DataFrame dei file CSV (un DataFrame per ogni CSV), poi successivamente viene specificato il percorso della cartella (path). A questo punto per automatizzare al meglio il processo di unione dei file CSV in unico DataFrame viene usato un ciclo for con una condizione al suo interno. Nel dettaglio il ciclo for in questione crea una "lista" (non salvata in una variabile) con tutti i nomi dei file nella cartella con la funzione "os.listdir()", in modo che ad ogni iterazione l'indice "nomedelfile" assume un diverso nome di file. Prima che "nomedelfile" assume un nuovo valore, il ciclo for segue una condizione: se "nomedelfile" finisce con ".csv" allora quest'ultimo viene letto e aggiunto alla lista "listadataframes" creata all'inizio del codice appositamente.

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import os

# Crea una lista di DataFrame
listadataframes = []

# Specifica il percorso
percorsofilecartella = r"C:\Users\utente\Downloads"

for nomedelfile in os.listdir(percorsofilecartella):
    # Controlla se il file è un file csv
    if nomedelfile.endswith(".csv"):
        # Crea la variabile percorsofilecsv per ogni file
        percorsofilecsv = os.path.join(percorsofilecartella,
nomedelfile)

        # Leggi il csv e aggiungilo alla lista
        df = pd.read_csv(percorsofilecsv)
        listadataframes.append(df)

print(listadataframes)
```

```
[       #                   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]]
```

IL CONTEGGIO DEI DATI DI UN DATAFRAME

Analisi del numero di DataFrame nella lista "listadataframes"

Il comando qui sotto permette di sapere il numero di DataFrame presenti nella lista "listadataframes". Len è l'acronimo di "length" cioè lunghezza.

```
len(listadataframes)
```

```
1
```

Analisi delle Dimensioni del DataFrame 23 nella lista "listadataframe"

Il comando "len(listadataframes[23])" permette al programma di contare il numero di istanze (righe) all'interno del DataFrame n°23.

```
if len(listadataframes) >= 24:
    print(len(listadataframes[23]))
else:
    print("L'indice 23 NON è presente nella lista listadataframe")
```

```
L'indice 23 NON è presente nella lista listadataframe
```

Per poter continuare questa esercitazione, è necessario installare la libreria "sklearn" attraverso il comando qui sotto.

```
pip install sklearn
```

```
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Preparing metadata (setup.py): started
```

```
  Preparing metadata (setup.py): finished with status 'error'
Note: you may need to restart the kernel to use updated packages.

  error: subprocess-exited-with-error

  python setup.py egg_info did not run successfully.
  exit code: 1

  [15 lines of output]
  The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
  rather than 'sklearn' for pip commands.

  Here is how to fix this error in the main use cases:
  - use 'pip install scikit-learn' rather than 'pip install sklearn'
  - replace 'sklearn' by 'scikit-learn' in your pip requirements files
    (requirements.txt, setup.py, setup.cfg, Pipfile, etc ...)
  - if the 'sklearn' package is used by one of your dependencies,
    it would be great if you take some time to track which package
uses
    'sklearn' instead of 'scikit-learn' and report it to their issue
tracker
  - as a last resort, set the environment variable
    SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid
this error

  More information is available at
  https://github.com/scikit-learn/sklearn-pypi-package
  [end of output]

  note: This error originates from a subprocess, and is likely not a
problem with pip.
error: metadata-generation-failed

Encountered error while generating package metadata.

See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

LO SPLITTING DATASET NORMALE E LE VISUALIZZAZIONI DEI DATI IN GRAFICI

Creazione e Suddivisione di un DataSet per la Predizione del Peso basato sull'Altezza

Il codice illustrato rappresenta un esempio pratico di "Dataset Splitting", un passo essenziale nella preparazione dei dati per l'addestramento e la valutazione dei modelli. Inizialmente, vengono importate le librerie necessarie, con un focus sull'utilizzo di "from" e "import" nella libreria "sklearn". Il concetto di "random seed" è introdotto per garantire la coerenza nei dati casuali. Successivamente, sono generati dati casuali sulle altezze e sui pesi attraverso distribuzioni gaussiane. I dati sono poi combinati in un unico DataSet. La suddivisione del DataSet in Training Set (70%) e Test Set (30%) è fondamentale per addestrare il modello e valutarne le prestazioni su dati indipendenti. Infine, vengono stampate le dimensioni dei dati di Training e di Test, fornendo una visione sintetica della suddivisione effettuata.

```
import numpy as np
from sklearn.model_selection import train_test_split # in questo caso viene solo importata una parte di libreria poichè è strettamente necessaria quella determinata funzione

# Creare dati casuali per altezze: variabile indipendente = input (cioè quello che serve per fare delle previsioni) e sono le Feature nel DataSet
# Pesi sono la variabile dipendente = output o target (cioè ciò che bisogna prevedere) del DataSet
np.random.seed(0)
altezze = np.random.normal(160, 10, 100) # la variabile "altezze" è indipendente, cioè non ha una formula in cui viene denominata un'altra variabile
# Formula di esempio:
pesi = 0.5 * altezze + np.random.normal(0, 5, 100) # la variabile "pesi" è dipendente, cioè ha una formula in cui viene denominata un'altra variabile

# Previsione del modello: dal valore delle altezze prevedere il peso

# Suddividere il dataset in training set (70%) e test set (30%) formando due DataSet
X_train, X_test, y_train, y_test = train_test_split(altezze, pesi, test_size=0.3, random_state=42) # riprendendo la formula di prima: le X sono i valori delle altezze perchè sono le Feature del DataSet, cioè l'input. Invece le Y sono gli output o target del DataSet, cioè i valori dei pesi. "test_size=0.3" vuol dire che il DataSet di Test è il 30% di quello totale mentre random_state sceglie in modo randomico i valori del DataSet per il Training e il Test
# Stampare le dimensioni dei training set e test set
```

```
print("Dimensioni del Training Set (altezze e pesi):", X_train.shape,
y_train.shape) # shape = dimensione dei DataSet di Training
print("Dimensioni del Test Set (altezze e pesi):", X_test.shape,
y_test.shape) # shape = dimensione dei DataSet di Test
```

```
Dimensioni del Training Set (altezze e pesi): (70,) (70,)
Dimensioni del Test Set (altezze e pesi): (30,) (30,)
```

Visualizzazione e Analisi delle Altezze tramite NumPy

Questo output particolare qui sotto non è altro che l'array di tutti i 100 valori che rappresentano altezze generate casualmente secondo la distribuzione Gaussiana nel codice di prima, con una media di 160 cm e una deviazione standard di 10 cm. Ogni valore dell'array corrisponde all'altezza di una persona esempio nel dataset. Inoltre nel codice per fare la "prova del 9" in fondo alla stampa dell'array è stato fatto stampare il numero totale di elementi dell'array tramite il comando "shape" che infatti è pari a 100, come descritto all'inizio.

Il comando "shape" stampa sempre un valore all'interno di una parentesi tonda insieme ad una virgola posta sempre alla fine del numero prima della chiusura della parentesi finale. Questo perchè a dir la verità il comando "shape" è stato progettato per le matrici e non per gli array. Le matrici sono degli array ma che si sviluppano non solo in larghezza (asse x) ma anche in altezza (asse y), definendo così più valori contemporaneamente. Si può e solitamente si usa questo comando anche per gli array perchè funziona correttamente, con il fatto però che quando Python non trova i dati dell'asse y lascierà uno spazio vuoto mantenendo così la virgola. Gli array si possono quindi definire unidimensionali mentre le matrici bidimensionali.

```
# Stampare l'array delle altezze
print("Array delle altezze:")
print(altezze)
# Dimensione dell'array
dimensione = altezze.shape
print("La grandezza dell'array delle altezze è di:", dimensione)
```

```
Array delle altezze:
[177.64052346 164.00157208 169.78737984 182.40893199 178.6755799
 150.2272212  169.50088418 158.48642792 158.96781148 164.10598502
 161.44043571 174.54273507 167.61037725 161.21675016 164.43863233
 163.33674327 174.94079073 157.94841736 163.13067702 151.45904261
 134.47010184 166.53618595 168.64436199 152.5783498  182.69754624
 145.45634325 160.45758517 158.1281615  175.32779214 174.6935877
 161.54947426 163.7816252  151.12214252 140.19203532 156.52087851
 161.56348969 172.30290681 172.02379849 156.12673183 156.97697249
 149.51447035 145.79982063 142.93729809 179.50775395 154.90347818
 155.61925698 147.4720464  167.77490356 143.86102152 157.8725972
 151.04533439 163.86902498 154.89194862 148.19367816 159.71817772
 164.28331871 160.66517222 163.02471898 153.65677906 156.37258834
 153.27539552 156.40446838 151.86853718 142.73717398 161.77426142
 155.98219064 143.69801653 164.62782256 150.92701636 160.51945396
 167.29090562 161.28982911 171.39400685 147.6517418  164.02341641
```

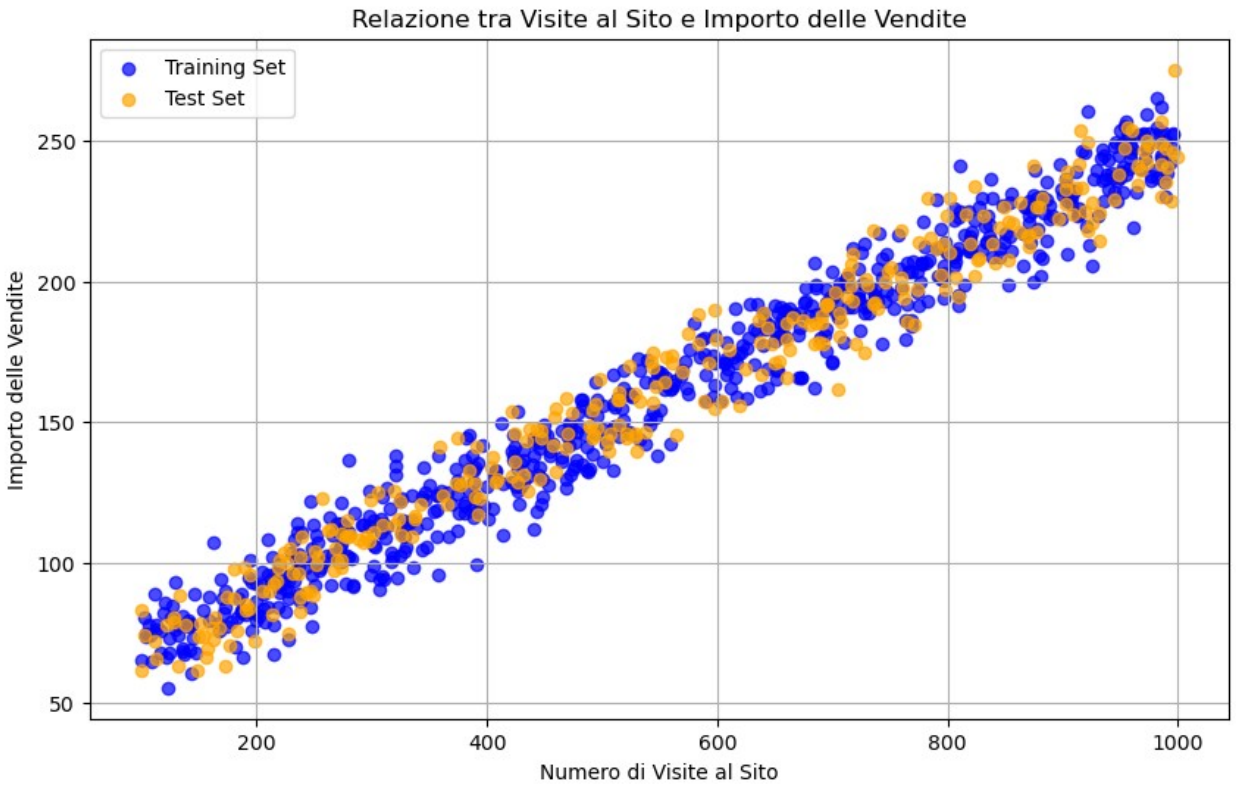
```
 153.15189909 151.29202851 154.21150335 156.88447468 160.56165342
 148.34850159 169.00826487 164.6566244  144.63756314 174.88252194
 178.95889176 171.78779571 158.20075164 149.29247378 170.54451727
 155.96823053 172.2244507  162.08274978 169.76639036 163.56366397
 167.06573168 160.10500021 177.85870494 161.26912093 164.01989363]
La grandezza dell'array delle altezze è di: (100,)
```

Analisi della Relazione tra Visite al Sito e Importo delle Vendite con Creazione e Suddivisione del DataSet

Il codice seguente è un altro esempio di Dataset Splitting, con una struttura sostanzialmente simile al codice precedente per quanto riguarda la generazione dei dati e la suddivisione del Dataset. L'elemento distintivo in questo codice è la rappresentazione grafica finale della relazione tra le visite al sito e l'importo delle vendite. Si evidenzia chiaramente una relazione lineare, mostrando una proporzionalità diretta tra la variabile dipendente (visite al sito) e quella indipendente (importo delle vendite). L'aumento delle visite al sito corrisponde a un aumento delle vendite. Tuttavia, si nota una leggera deviazione dalla linearità, attribuibile al rumore introdotto dalla distribuzione gaussiana nella formula finale dell'importo delle vendite. Il rumore rappresenta variazioni impreviste o interferenze nei dati, generando un andamento non perfettamente lineare e aggiungendo una componente casuale attraverso "np.random.normal()".

```
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
# Creare dati casuali per le visite al sito: variabile indipendente =
input (cioè quello che serve per fare delle previsioni) e sono le
Feature nel DataSet
# L'importo delle vendite sono la variabile dipendente = output o
target (cioè ciò che bisogna prevedere) del DataSet
np.random.seed(0)
visite_al_sito = np.random.randint(100, 1000, 1000) # la variabile
"visite al sito" è indipendente, cioè non ha una formula in cui viene
denominata un'altra variabile. "np.random.randint(100, 1000, 1000)"
vuol dire che vengono creati dei valori randomici, sempre attraverso
la libreria numpy. Si legge: il primo parametro (100) indica il valore
minimo che può assumere il numero mentre il secondo parametro (1000)
indica il valore massimo, infine il terzo parametro (1000) indica il
numero di valori da generare
importo_vendite = 50 + 0.2 * visite_al_sito + np.random.normal(0, 10,
1000) # la variabile "importo delle vendite" è dipendente, cioè ha una
formula in cui viene denominata un'altra variabile
# 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) # riprendendo la
```

```
formula di prima: le X sono i valori delle visite al sito perchè sono le Feature del DataSet, cioè l'input. Invece le Y sono gli output o target del DataSet, cioè i valori degli importi vendite. "test_size=0.3" vuol dire che il DataSet di Test è il 30% di quello totale mentre random_state sceglie in modo randomico i valori del DataSet per il Training e il Test

# Previsione del modello: dal valore delle visite al sito prevedere l'importo vendite

# Creazione di un grafico a dispersione
plt.figure(figsize=(10, 6)) # dimensione del grafico
plt.scatter(X_train, y_train, label='Training Set', color='blue', alpha=0.7) # label è il nome della legenda, alpha è il valore della trasparenza: più è vicino ad 0 come valore i pallini del grafico saranno più trasparenti
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)
```



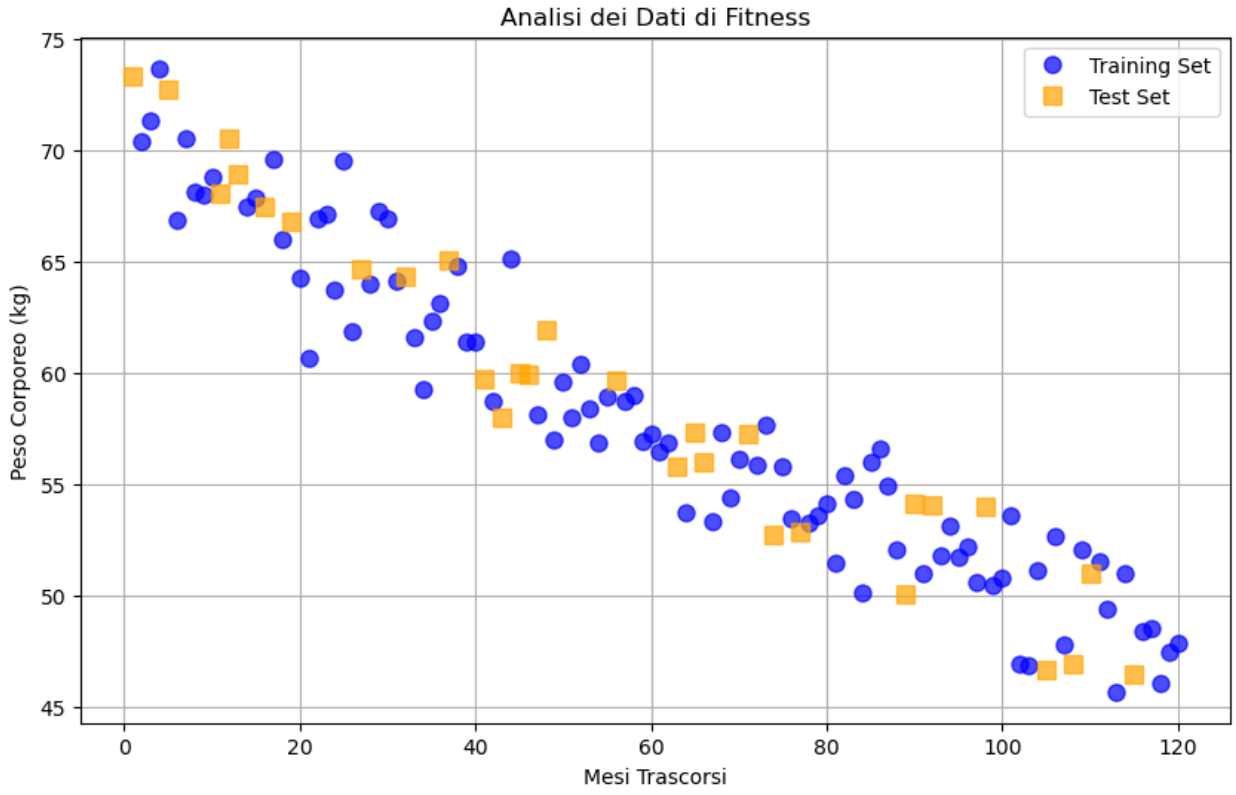

```
Dimensioni del Training Set (visite al sito e importo delle vendite): (700,) (700,)
Dimensioni del Test Set (visite al sito e importo delle vendite): (300,) (300,)
```

Analisi dei Dati di Fitness con Predizione del Peso Corporeo

Il codice sottostante è un'altro esempio di Dataset Splitting. La struttura è perlopiù identica al codice precedente riguardo la creazione dei dati e il Dataset Splitting. L'unica differenza con il codice precedente è il grafico finale sulla relazione tra i mesi trascorsi e peso corporeo. Da come si può notare, c'è una relazione sempre di tipo lineare ma ben diversa dal codice precedente. Infatti si può subito notare come ci sia più o meno una proporzionalità diretta ma negativa, quindi non inversa come si può pensare. Rispetto al codice di prima l'output viene stampato in maniera decrescente ma è pur sempre lineare, infatti non viene un'iperbole. Quindi il grafico va letto nel modo in cui all'aumentare della variabile indipendente (mesi trascorsi) diminuisce la variabile dipendente (peso corporeo). Il fatto che non sia perfettamente lineare è dovuto al rumore, generato dalla gaussiana nella formula finale (cioè quella del peso corporeo).

```
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
```

```
# Creare dati casuali per i mesi trascorsi: variabile indipendente = input (cioè quello che serve per fare delle previsioni) e sono le Feature nel DataSet
# Il peso corporeo è la variabile dipendente = output o target (cioè ciò che bisogna prevedere) del DataSet
np.random.seed(0)
n=120 # è un parametro, non è una variabile ed è molto comodo in quanto se si cambia quello si cambia tutto ciò "collegato" ad esso
mesi_trascorsi = np.arange(1, n+1) # la variabile "visite al sito" è indipendente, cioè non ha una formula in cui viene denominata un'altra variabile. "np.arange(1, n+1)" serve per generare un array di numeri interi da 1 a n, inclusi, che rappresentano i singoli mesi nel periodo di osservazione.
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) # riprendendo la formula di prima: le X sono i valori dei mesi trascorsi perchè sono le Feature del DataSet, cioè l'input. Invece le Y sono gli output o target del DataSet, cioè i valori del peso corporeo. "test_size=0.25" vuol dire che il DataSet di Test è il 25% di quello totale mentre random_state sceglie in modo randomico i valori del DataSet per il Training e il Test

# Creazione di un grafico a linee
plt.figure(figsize=(10, 6)) #dimesioni del grafico
plt.plot(X_train, y_train, label='Training Set', marker='o', color='blue', linestyle='', markersize=8,alpha=0.7) # label è il nome della legenda, alpha è il valore della trasparenza: più è vicino ad 0 come valore i pallini del grafico saranno più trasparenti
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)
```



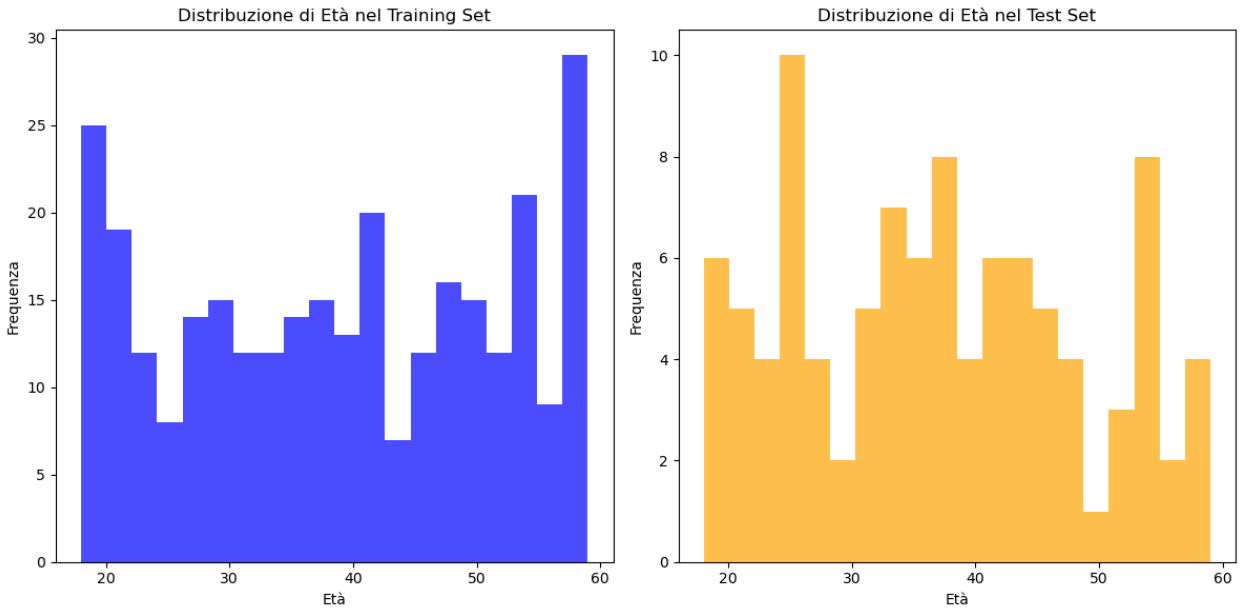

```
Dimensioni del Training Set (mesi trascorsi e peso corporeo): (90,)
(90,)
Dimensioni del Test Set (mesi trascorsi e peso corporeo): (30,) (30,)
```

Confronto delle Distribuzioni di Età tra Training Set e Test Set

Nel seguente codice, vengono direttamente creati due DataSet: uno di Training e uno di Test. Nella fase iniziale, entrambi i dataset vengono popolati con valori casuali. Successivamente, nella seconda parte del codice, vengono generati due grafici distinti. Il grafico blu rappresenta la distribuzione delle età nel Training Set, mentre il grafico arancione rappresenta la distribuzione delle età nel Test Set. Per la creazione dei grafici, si fa uso della libreria "matplotlib".

```
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) #
"np.random.randint(100, 1000, 1000)" vuol dire che vengono creati dei
valori randomici, sempre attraverso la libreria numpy. Si legge: il
primo parametro (100) indica il valore minimo che può assumere il
numero mentre il secondo parametro (1000) indica il valore massimo,
```

```
infine il terzo parametro (1000) indica il numero di valori da
generare
eta_test_set = np.random.randint(18, 60, 100)
# Confronto delle distribuzioni di età
# Primo grafico
plt.figure(figsize=(12, 6)) # dimesioni del grafico
plt.subplot(1, 2, 1) # permette di creare all'interno di una figura
più plot. Il primo valore indica il numero di righe, il secondo indica
il numero di colonne ed invece il terzo indica in quale colonna
stampare
plt.hist(eta_training_set, bins=20, color='blue', alpha=0.7) # crea un
istogramma, bins vuole indicare la barre verticali cioè i cosidetti
"bins"
plt.title('Distribuzione di Età nel Training Set')
plt.xlabel('Età')
plt.ylabel('Frequenza')
# Secondo grafico
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()
```



LO SPLITTING DATASET CON LE CLASSI E LE VISUALIZZAZIONI DEI DATI IN GRAFICI

Esempio di Split Stratificato in un Dataset con Classi Bilanciate

Questo codice, simile a quelli precedentemente esaminati, impiega librerie come numpy e sklearn per generare casualmente valori di x e y. Un elemento distintivo è l'introduzione del concetto di classi, identificate nelle etichette dell'array y come A e B, simboleggianti categorie come "cane" o "gatto". La novità risiede nell'adozione di uno "split stratificato", garantendo che la proporzione delle categorie sia mantenuta sia nel Training che nel Test. Questo approccio ottimizza l'apprendimento del modello, consentendogli di imparare equamente da entrambe le classi e migliorare la capacità di generalizzazione e precisione su nuovi dati. I risultati stampati riflettono questa equità, dimostrando un mantenimento approssimativo delle proporzioni tra i due set.

```
from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(3)
# Supponiamo di avere un dataset con feature (input) X e target
(output) y
X = np.random.rand(100, 2) # crea dati del dataset (100 campioni, 2
feature)
y = np.random.choice(['A', 'B'], size=100) # etichette (stringhe A e
B) di classi casuali (categorie), size indica la grandezza di y (cioè
ha 100 valori)
# Per mostrare che i valori siano casuali, essi vengono stampati
print("I valori di x sono:")
print(X)
print("I valori di y sono:")
print(y)

# Calcolare le percentuali delle classi nell'intero dataset originale
percentuale_classe_A = sum(y == 'A') / len(y)*100 # nel primo caso la
percentuale viene calcolata dividendo il numero dei valori di A per la
lunghezza di y
percentuale_classe_B = 100 - percentuale_classe_A # nel secondo per
calcolare la percentuale mancante si sottrae a 100 il valore della
percentuale 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) # "test_size=0.3" vuol dire che il
DataSet di Test è il 30% di quello totale mentre random_state sceglie
in modo randomico i valori del DataSet per il Training e il Test
```

```python
# Calcolare le percentuali delle classi nel training set e nel test
set
percentuale_classe_A_train = sum(y_train == 'A') / len(y_train)*100
percentuale_classe_B_train = 100 - percentuale_classe_A_train

percentuale_classe_A_test = sum(y_test == 'A') / len(y_test)*100
percentuale_classe_B_test = 100 - percentuale_classe_A_test

# Stampare delle proporzioni
print("Percentuale Classe A nel Data Set:", percentuale_classe_A)
print("Percentuale Classe B nel Data Set:", percentuale_classe_B)
print("Percentuale Classe A nel Training Set:",
percentuale_classe_A_train)
print("Percentuale Classe B nel Training Set:",
percentuale_classe_B_train)
print("Percentuale Classe A nel Test Set:", percentuale_classe_A_test)
print("Percentuale Classe B nel Test Set:", percentuale_classe_B_test)
```

```
I valori di x sono:
[[0.5507979  0.70814782]
 [0.29090474 0.51082761]
 [0.89294695 0.89629309]
 [0.12558531 0.20724288]
 [0.0514672  0.44080984]
 [0.02987621 0.45683322]
 [0.64914405 0.27848728]
 [0.6762549  0.59086282]
 [0.02398188 0.55885409]
 [0.25925245 0.4151012 ]
 [0.28352508 0.69313792]
 [0.44045372 0.15686774]
 [0.54464902 0.78031476]
 [0.30636353 0.22195788]
 [0.38797126 0.93638365]
 [0.97599542 0.67238368]
 [0.90283411 0.84575087]
 [0.37799404 0.09221701]
 [0.6534109  0.55784076]
 [0.36156476 0.2250545 ]
 [0.40651992 0.46894025]
 [0.26923558 0.29179277]
 [0.4576864  0.86053391]
 [0.5862529  0.28348786]
 [0.27797751 0.45462208]
 [0.20541034 0.20137871]
 [0.51403506 0.08722937]
 [0.48358553 0.36217621]
 [0.70768662 0.74674622]
 [0.69109292 0.68918041]
 [0.37360012 0.6681348 ]
```

```
[0.33984866 0.57279387]
[0.32580716 0.44514505]
[0.06152893 0.24267542]
[0.97160261 0.2305842 ]
[0.69147751 0.65047686]
[0.72393914 0.47508861]
[0.59666377 0.06696942]
[0.07256214 0.19897603]
[0.151861   0.10010434]
[0.12929386 0.55327773]
[0.18781482 0.95210124]
[0.68161178 0.54101967]
[0.7071806  0.26388667]
[0.92672568 0.83919306]
[0.7263195  0.48023996]
[0.84210319 0.74475232]
[0.66032591 0.91397527]
[0.63366556 0.36594058]
[0.55284457 0.19638058]
[0.1920723  0.72566962]
[0.7849367  0.97209836]
[0.85097142 0.54359433]
[0.08979087 0.48887324]
[0.92793635 0.7876182 ]
[0.48509423 0.45527936]
[0.21798577 0.17721338]
[0.07362367 0.89239319]
[0.64017662 0.14333232]
[0.41412692 0.04910892]
[0.20937335 0.73070812]
[0.65112277 0.4789783 ]
[0.27478051 0.65222313]
[0.95644951 0.43552056]
[0.07013251 0.05773149]
[0.08287102 0.95970719]
[0.54076084 0.83746243]
[0.17003354 0.26034507]
[0.69197751 0.89557033]
[0.34068848 0.0646732 ]
[0.86411967 0.29087245]
[0.74108241 0.15803365]
[0.69496344 0.84141962]
[0.72715208 0.35910752]
[0.72668975 0.13946712]
[0.31381912 0.41958276]
[0.87721204 0.15374021]
[0.88012479 0.79896432]
[0.9716243  0.36770298]
[0.20493977 0.24057032]
```

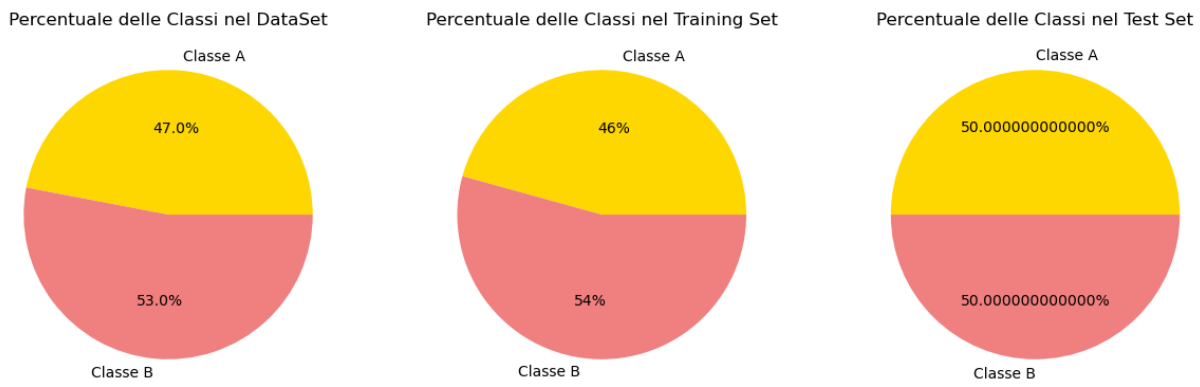
```
 [0.8278628  0.96522815]
 [0.69881    0.48249704]
 [0.28704976 0.83368788]
 [0.87217951 0.09213159]
 [0.21594947 0.83176109]
 [0.8483039  0.314653  ]
 [0.2792946  0.43081502]
 [0.5394465  0.09556682]
 [0.83691214 0.53473487]
 [0.77496782 0.23083627]
 [0.96529335 0.75102731]
 [0.34309386 0.94852765]
 [0.70051178 0.84056109]
 [0.04549731 0.05564154]
 [0.74273727 0.30468643]
 [0.51678437 0.15626242]
 [0.97795241 0.50275105]
 [0.82900108 0.0740378 ]
 [0.47891545 0.06227948]
 [0.88424143 0.44581018]]
I valori di y sono:
['B' 'B' 'A' 'B' 'B' 'B' 'A' 'B' 'B' 'B' 'B' 'B' 'A' 'A' 'B' 'A' 'A'
'B'
 'A' 'B' 'B' 'B' 'B' 'A' 'A' 'B' 'A' 'A' 'B' 'B' 'A' 'B' 'B' 'A' 'A'
'A'
 'B' 'A' 'A' 'B' 'A' 'B' 'B' 'B' 'A' 'A' 'B' 'B' 'A' 'A' 'B' 'A' 'B'
'B'
 'B' 'A' 'A' 'B' 'B' 'B' 'A' 'A' 'B' 'A' 'A' 'A' 'A' 'A' 'B' 'B' 'A'
'A'
 'A' 'B' 'A' 'A' 'A' 'B' 'A' 'B' 'B' 'A' 'A' 'B' 'B' 'B' 'B' 'B' 'B'
'B'
 'A' 'B' 'B' 'B' 'A' 'B' 'A' 'A' 'A' 'A']
Percentuale Classe A nel Data Set: 47.0
Percentuale Classe B nel Data Set: 53.0
Percentuale Classe A nel Training Set: 45.714285714285715
Percentuale Classe B nel Training Set: 54.285714285714285
Percentuale Classe A nel Test Set: 50.0
Percentuale Classe B nel Test Set: 50.0
```

Visualizzazione della Distribuzione Percentuale delle Classi in Diverse Partizioni del Datasetv

Nel grafico sottostante vengono stampati la percentuale di classi del DataSet, la percentuale di classi del Training Set, la percentuale di classi del Test Set.

```python
import matplotlib.pyplot as plt
# Etichette delle classi
labels=["Classe A", "Classe B"]
```

```
# Colori delle fette del grafico
colors = ['gold', 'lightcoral']
# Crea un grafico a torta con etichette
plt.figure(figsize=(15,10)) # dimesioni del grafico
plt.subplot(1,3,1) # permette di creare all'interno di una figura più
plot. Il primo valore indica il numero di righe, il secondo indica il
numero di colonne ed invece il terzo indica in quale colonna stampare
plt.pie([percentuale_classe_A, percentuale_classe_B], labels=labels,
colors=colors, autopct='%1.1f%%') # si usa pie perchè in questo caso è
un grafico a torta, autopct è l'etichetta con il valore: più numeri si
mettono dopo il punto e più valori decimali vengono mostrati
plt.title('Percentuale delle Classi nel DataSet')

# Crea un grafico a torta con etichette
plt.subplot(1,3,2)
plt.pie([percentuale_classe_A_train, percentuale_classe_B_train],
labels=labels, colors=colors, autopct='%1.0f%%')#si usa pie perchè in
questo caso è un grafico a torta, autopct è l'etichetta con il valore:
più numeri si mettono dopo il punto e più valori decimali vengono
mostrati
plt.title('Percentuale delle Classi nel Training Set')

# Crea un grafico a torta con etichette
plt.subplot(1,3,3)
plt.pie([percentuale_classe_A_test, percentuale_classe_B_test],
labels=labels, colors=colors, autopct='%1.12f%%')#si usa pie perchè in
questo caso è un grafico a torta, autopct è l'etichetta con il valore:
più numeri si mettono dopo il punto e più valori decimali vengono
mostrati
plt.title('Percentuale delle Classi nel Test Set')
plt.show()
```



Analisi statistica di un dataset: Estrazione casuale di campione, Calcolo della media e deviazione standard

Il codice qui sotto produce un ampio dataset contenente 24 milioni di numeri casuali nell'intervallo da 0 a 100. Successivamente, viene estratto casualmente un campione che rappresenta circa il 30% del totale. Vengono poi calcolate la media e la deviazione standard di questo campione, dove la media rappresenta il valore medio dei numeri nel campione e la deviazione standard quantifica quanto i numeri si discostano dalla media. Infine, il codice calcola anche la media e la deviazione standard per l'intero dataset, riflettendo il valore medio e la dispersione dei numeri nell'insieme completo. I risultati stampati includono la media e la deviazione standard del campione casuale, oltre alla media e deviazione standard dell'intero dataset.

```
import random
import numpy as np

dataset=[]
# Creazione di un dataset di 24 milioni elementi (ad esempio, dati casuali)
popolazione=24000000
for i in range(popolazione):
    dataset.append(random.randint(0, 100000)) # il Dataset prima del ciclo for è vuoto ma viene definito il numero della popolazione, dopo bisogna però aggiungere ogni singolo elemento di 24000000 alla lista creandolo in maniera randomica e per far questo utiizza un ciclo for

# Estrazione di un campione casuale e non: cioè una parte del Dataset creando così un sub DataSet
campione = int(round(0.3 * popolazione)) # per creare un sub DataSet in maniera del tutto causale, viene estratta casualmente una parte del Dataset utilizzando questa formula. Lo 0.3 = 3%. Round = approssima all'intero e poi Int trasforma il risultato in intero
campione_casuale = random.sample(dataset, campione) # il comando random.sample() campiona (estrae) il DataSet, si usa "sample" quando bisogna estrarre casualmente un valore. "dataset" rappresenta la sequenza dalla quale estrarre un campione casuale invece "campione" rappresenta il numero di elementi da estrarre casualmente dal dataset (il 30% perchè è collegato alla formula di sopra).

# Calcolo della media e della deviazione standard del campione
media_campione = np.mean(campione_casuale) # il comando np.mean() calcola la media del campione casuale
deviazione_standard_campione = np.std(campione_casuale) # calcola la deviazione standard (std = standard deviation). La deviazione standard è una misura di dispersione o variabilità dei dati in un insieme. In sostanza, dice quanto i dati sono distanziati dalla media.

# 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}") # il ".2f"
indica quanti valori dopo la virgola mostrare, più è alto più vengono
mostrati
print(f"Deviazione standard del campione casuale:
{deviazione_standard_campione: .2f}")
print(f"Media del dataset completo: {media_dataset: .10f}")
print(f"Deviazione standard del dataset completo:
{deviazione_standard_dataset: .10f}")
```

```
Media del campione casuale:  49999.39
Deviazione standard del campione casuale:  28866.23
Media del dataset completo:  50000.6593462917
Deviazione standard del dataset completo:  28865.7966476865
```

Generazione di DataFrame con Distribuzione Personalizzata per la Colonna 'ColonnaAB

Il codice utilizza la libreria pandas per creare un DataFrame con una colonna chiamata "ColonnaAB", seguendo una distribuzione specifica tra le categorie 'A' e 'B'. Imposta un seed per la riproducibilità, definisce il numero totale di elementi nel DataFrame (100.000) e la percentuale di elementi della categoria 'A' (0.7). Utilizza la funzione np.random.choice per generare la colonna con la distribuzione desiderata tra le categorie, specificata attraverso il parametro p. Infine, crea il DataFrame con pandas e lo visualizza.

```
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
```

```
      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]
```

Creazione di Tre Subset Equamente Dimensionati da un DataFrame

Il codice crea tre sottoinsiemi (subset) simili dal DataFrame originale. Inizia con "subset1", estrae casualmente un terzo dei dati dal DataFrame e li rimuove. Poi, "subset2" estrae la metà dei dati rimasti e li elimina. Infine, "subset3" contiene i dati residui nel DataFrame dopo "subset1" e "subset2".

```
# 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
```

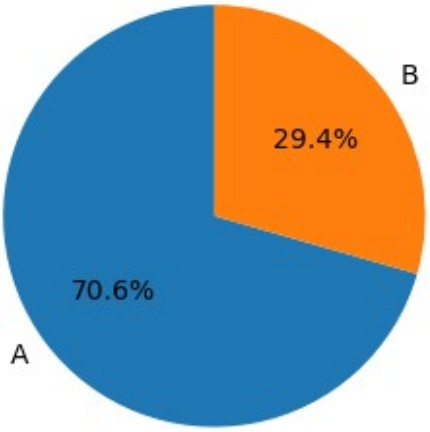
Visualizzazione delle Percentuali di Distribuzione di A e B nei Subsets

Qua sotto vengono stampati i grafici esplicativi:

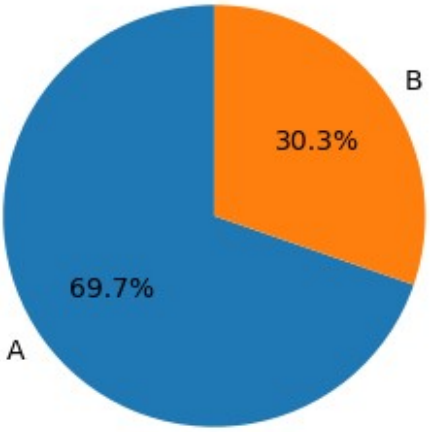
```
# Calcolare le percentuali di "A" e "B" per ogni subset
percentuali_subset1 =
subset1['ColonnaAB'].value_counts(normalize=True)
percentuali_subset2 =
subset2['ColonnaAB'].value_counts(normalize=True)
percentuali_subset3 =
subset3['ColonnaAB'].value_counts(normalize=True)

# Creare i grafici a torta
fig, axs = plt.subplots(3, 1, figsize=(6, 12))

# Subset 1
axs[0].pie(percentuali_subset1, labels=percentuali_subset1.index,
```

```
autopct='%1.1f%%', startangle=90)
axs[0].set_title('Subset 1')

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

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

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

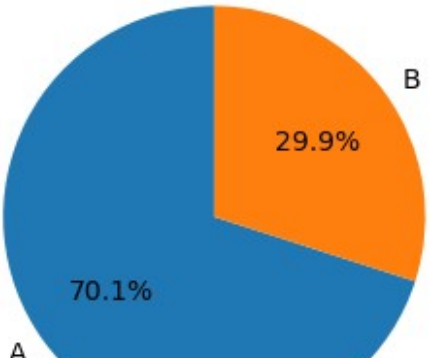
Subset 1



Subset 2



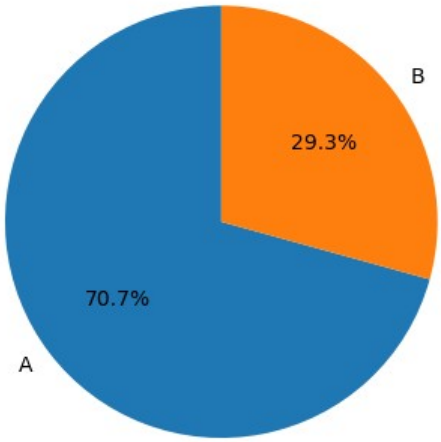
Subset 3



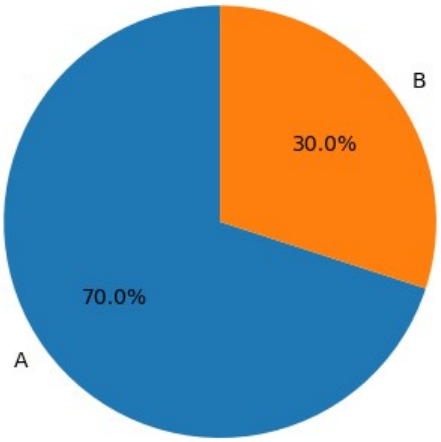
Confronto delle Distribuzioni di Classi tra Training e Test Sets per Tre Subset

```
# 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()
```

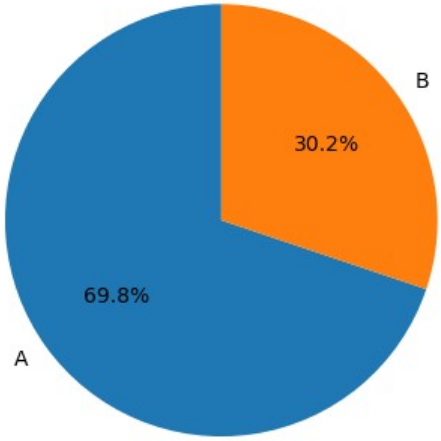
Train Subset 1



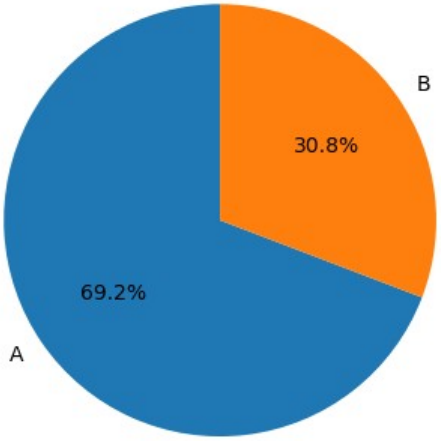
Test Subset 1



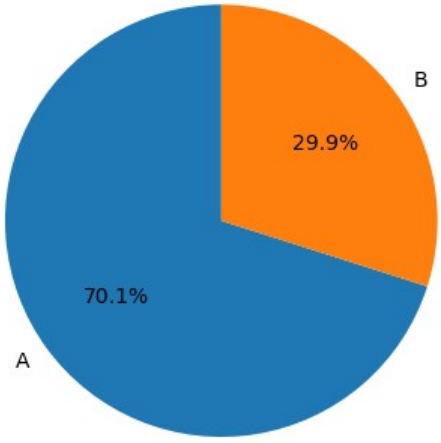
Train Subset 2



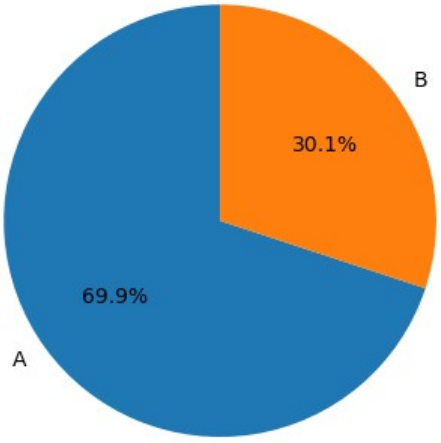
Test Subset 2



Train Subset 3



Test Subset 3

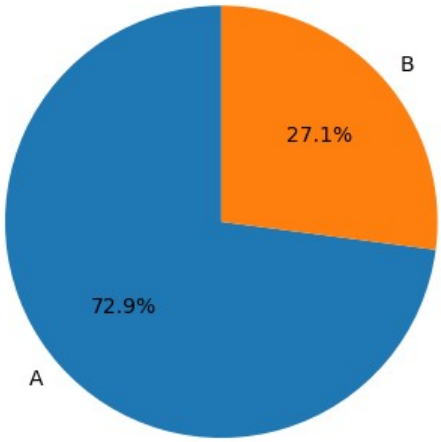


Analisi della Distribuzione Train-Test in Tre Subset Dati con Grafico a Torta

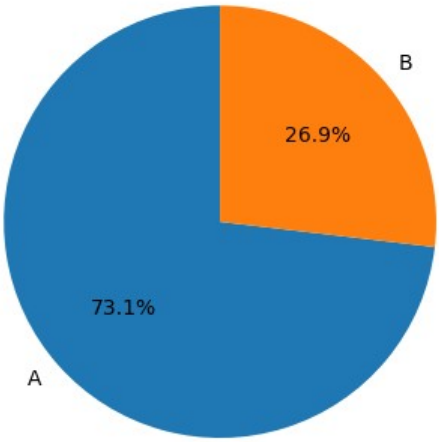
```
np.random.seed(41)

# 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()
```

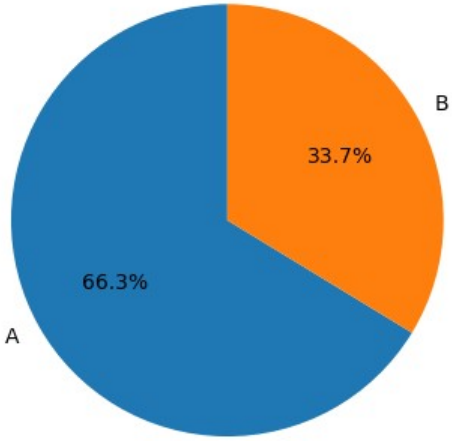
Train Subset 1



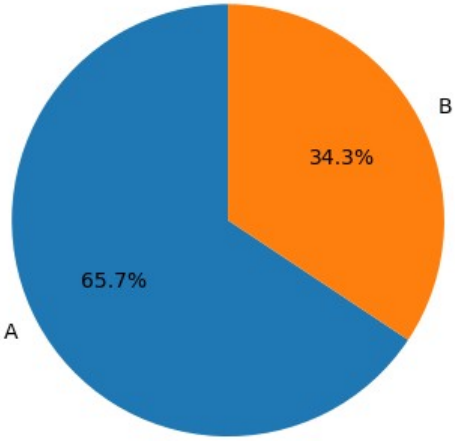
Test Subset 1



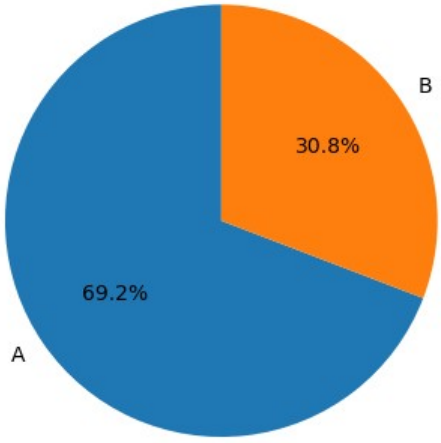
Train Subset 2



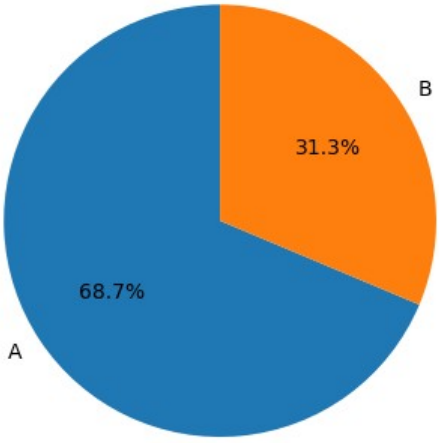
Test Subset 2



Train Subset 3



Test Subset 3

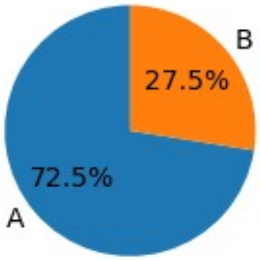


Visualizzazione dei Training Set e Test Set in Suddivisioni Equilibrate: Analisi delle Proporzioni di Classi tra i Subset

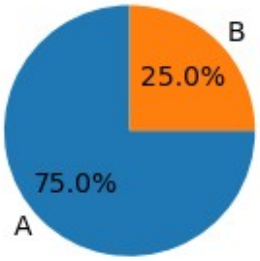
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

# 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()
```

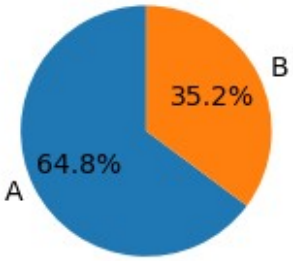
Train Subset 1



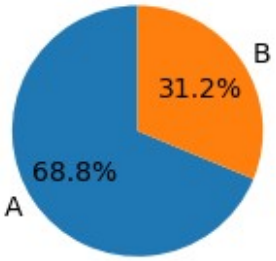
Test Subset 1



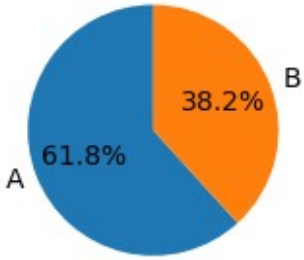
Train Subset 2



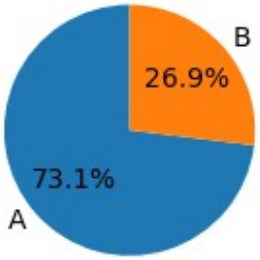
Test Subset 2



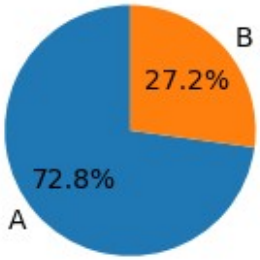
Train Subset 3



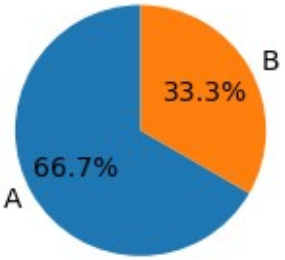
Test Subset 3



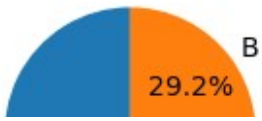
Train Subset 4



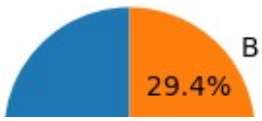
Test Subset 4



Train Subset 5



Test Subset 5



LA DEVIAZIONE STANDARD

Calcolo della Deviazione Standard in Python

La funzione calcola la deviazione standard di una lista di numeri utilizzando la formula matematica. Inizia calcolando la media dei numeri e successivamente la somma dei quadrati delle differenze tra ciascun numero e la media. Infine, divide questa somma per il numero totale di elementi e ne calcola la radice quadrata. La deviazione standard misura la dispersione dei dati intorno alla media. Un esempio di utilizzo con la lista [1, 2, 3, 4, 5] viene mostrato nel codice.

```
def calcola_deviazione_standard(lista):
    n = len(lista)

    # Calcola la media
    media = sum(lista) / n

    # Calcola la somma dei quadrati delle differenze dalla media
    somma_quadrati_diff = sum((x - media) ** 2 for x in lista)

    # Calcola la deviazione standard
    deviazione_standard = (somma_quadrati_diff / n) ** 0.5

    return deviazione_standard

# Esempio di utilizzo
numero_lista = [1, 2, 3, 4, 5]
deviazione_standard = calcola_deviazione_standard(numero_lista)

    # la formula è: σ = √(Σ(xi - x¯)² / n)
    # √ = radice quadrata
    # Σ = sommatoria di tutti gli elementi dentro la parentesi quadra
    # xi = sono i singoli valori dei dati
    # x¯ = è la media dei dati
    # n = è il numero totale di dati

# Stampa il risultato
print(f"La deviazione standard della lista è: {deviazione_standard}")
```

```
La deviazione standard della lista è: 1.4142135623730951
```

Visualizzazione degli Outliers in un DataFrame con Calcolo di Media e Deviazione Standard

Il codice usa la libreria pandas per generare un DataFrame denominato df, che include una colonna chiamata 'Valori' con una serie di valori. Successivamente, calcola la media e la deviazione standard della colonna 'Valori' tramite i metodi mean() e std(). Infine, visualizza la deviazione standard ottenuta.

```
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.pyplot as plt

# Crea un DataFrame di esempio
data = {'Valori': [1, 2, 3, 4, 5, 10, 15, 20, 25, 300, 1000,
100000000, -50000000, -50]}
df = pd.DataFrame(data)
# Lista con outliers da entrambi i lati

# Calcola la media e la deviazione standard
mean_value = df['Valori'].mean()
std_dev = df['Valori'].std()
std_dev
```

```
30786384.39895254
```

Rilevamento degli Outliers tramite Criterio dei +3 Sigma dalla Media

Il codice individua outliers nel DataFrame originale, selezionando valori oltre 3 deviazioni standard dalla media. Crea un DataFrame "outliers" con le righe di df in cui i valori di "Valori" superano o scendono sotto questa soglia. Infine, mostra il DataFrame "outliers". Questo metodo aiuta a identificare e rimuovere dati anomali che potrebbero influenzare negativamente analisi o modelli di machine learning.

```
#Identifica gli outliers consiederando +3 sigma dalla media
outliers=df[(df["Valori"]>mean_value+3*std_dev) |
(df["Valori"]<mean_value-3*std_dev)]
outliers
```

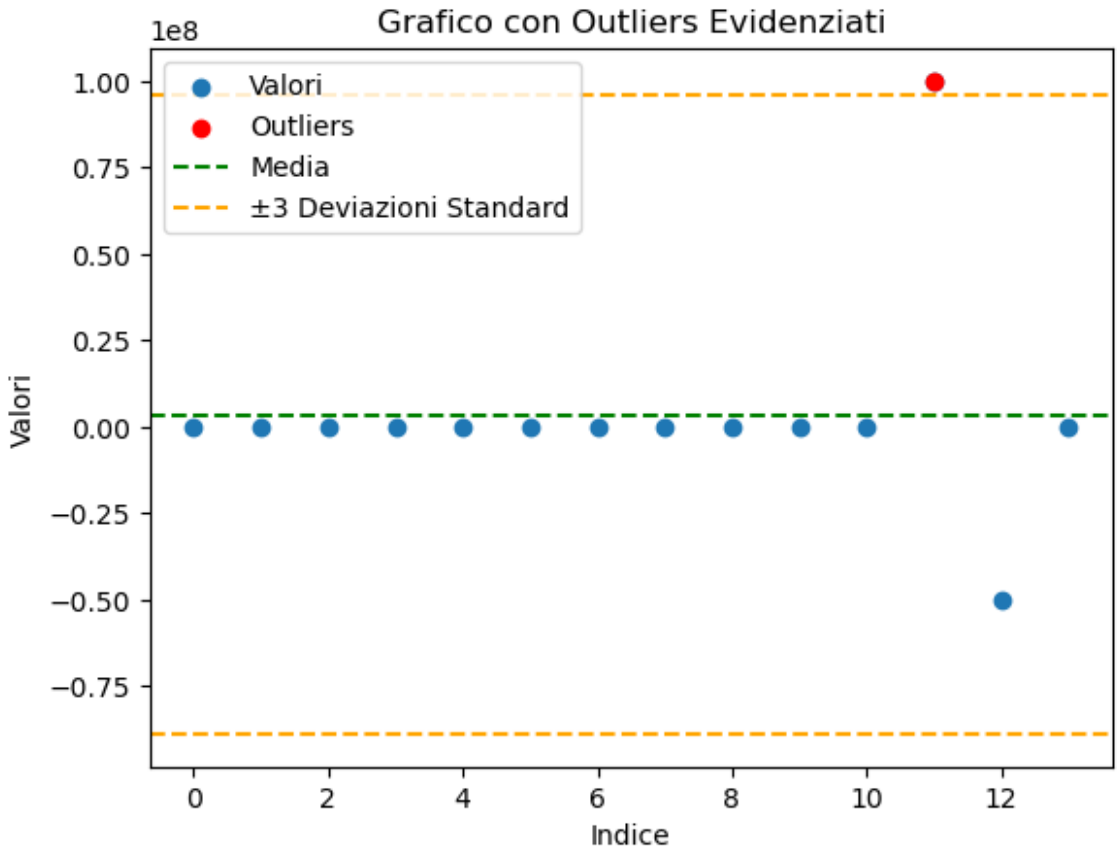
```
       Valori
11  100000000
```

Visualizzazione dei Valori con Evidenziazione degli Outliers

Il codice crea uno scatter plot con i valori del DataFrame df. Identifica gli outliers nel grafico, evidenziandoli in rosso e tracciando la media e le deviazioni standard. Aggiunge etichette agli assi, un titolo e una legenda.

```
# 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()
```



Rilevazione e Marcatura degli Outlier in un DataFrame Pandas con Analisi delle Features e Intervalli di Confidenza

Il codice importa le librerie pandas e matplotlib e crea un DataFrame di esempio con 4 features. Viene definito il numero minimo di features necessarie per considerare un dato un outlier, e si

stabilisce un intervallo di confidenza (k=3). Successivamente, il codice itera su ogni feature, calcola la media e la deviazione standard, e identifica gli outliers in base alla soglia definita. Gli indici degli outliers vengono salvati in una lista. Infine, il DataFrame viene ampliato con colonne aggiuntive che indicano se ciascun dato è un outlier per la rispettiva feature.

```
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
```

```
     Feature1  Feature2  Feature3    Feature4  Outlier_Feature1  \
0           1         2         5           1             False
1         200         4        10     -200000             False
2           3         6        15           3             False
3           4         8     20000  4000000000             False
4       50000        10        25           5             False
5          10        20        50          10             False
6          15        30        75          15             False
7          20        40       100          20             False
8     2500000       500       125         200             False
9   300000000        60       150          30             False
10  100000000       200    500000       10000             False

    Outlier_Feature2  Outlier_Feature3  Outlier_Feature4
```

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

Rilevazione e Rimozione degli Outliers: Identificazione e Eliminazione delle Righe con Caratteristiche Fuoriscala

```
#Elimina le righe corrispondenti agli outliers quelli che hanno una features fuoriscala
outliers = df['Num_Outliers'] = df.filter(like='Outlier_').sum(axis=1)
df
```

```
     Feature1  Feature2  Feature3    Feature4  Outlier_Feature1  \
0           1         2         5           1             False
1         200         4        10     -200000             False
2           3         6        15           3             False
3           4         8     20000  4000000000             False
4       50000        10        25           5             False
5          10        20        50          10             False
6          15        30        75          15             False
7          20        40       100          20             False
8     2500000       500       125         200             False
9   300000000        60       150          30             False
10  100000000       200    500000       10000             False

    Outlier_Feature2  Outlier_Feature3  Outlier_Feature4  Num_Outliers
0              False             False             False             0
1              False             False             False             0
2              False             False             False             0
3              False             False              True             1
4              False             False             False             0
5              False             False             False             0
6              False             False             False             0
```

```
7               False              False              False              0

8               False              False              False              0

9               False              False              False              0

10              False               True              False              1
```

Selezione delle righe con Numero Minimo di Features Superante la Soglia

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

```
     Feature1  Feature2  Feature3    Feature4  Outlier_Feature1  \
3           4         8     20000  4000000000             False
10  100000000       200    500000       10000             False

    Outlier_Feature2  Outlier_Feature3  Outlier_Feature4  Num_Outliers

3              False             False              True             1

10             False              True             False             1
```

Aggiunta della Colonna 'Is_Outlier' per Identificare gli Outlier nel DataFrame

```
# Aggiungi una colonna che indica se il record è un outlier o meno
df['Is_Outlier'] = df.index.isin(outliers.index)
df
```

```
     Feature1  Feature2  Feature3    Feature4  Outlier_Feature1  \
0           1         2         5           1             False
1         200         4        10     -200000             False
2           3         6        15           3             False
3           4         8     20000  4000000000             False
4       50000        10        25           5             False
5          10        20        50          10             False
6          15        30        75          15             False
7          20        40       100          20             False
8     2500000       500       125         200             False
9   300000000        60       150          30             False
10  100000000       200    500000       10000             False

    Outlier_Feature2  Outlier_Feature3  Outlier_Feature4  Num_Outliers
```

```
\
0              False              False              False              0
1              False              False              False              0
2              False              False              False              0
3              False              False               True              1
4              False              False              False              0
5              False              False              False              0
6              False              False              False              0
7              False              False              False              0
8              False              False              False              0
9              False              False              False              0
10             False               True              False              1


    Is_Outlier
0        False
1        False
2        False
3         True
4        False
5        False
6        False
7        False
8        False
9        False
10        True
```

Eliminazione Colonne Ausiliarie e Indicatori di Outlier dal DataFrame

```
# Rimuovi colonne ausiliarie
df.drop(df.filter(like='Outlier_').columns, axis=1, inplace=True)
df.drop('Num_Outliers', axis=1, inplace=True)
df
```

```
     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
```

Filtraggio dei dati: Esclusione degli Outliers da un DataFrame

```
df_filtered = df[df['Is_Outlier'] == False ]
df_filtered
```

```
    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
```

Selezionare dati senza valori anomali: Filtraggio del DataFrame per escludere gli outliers

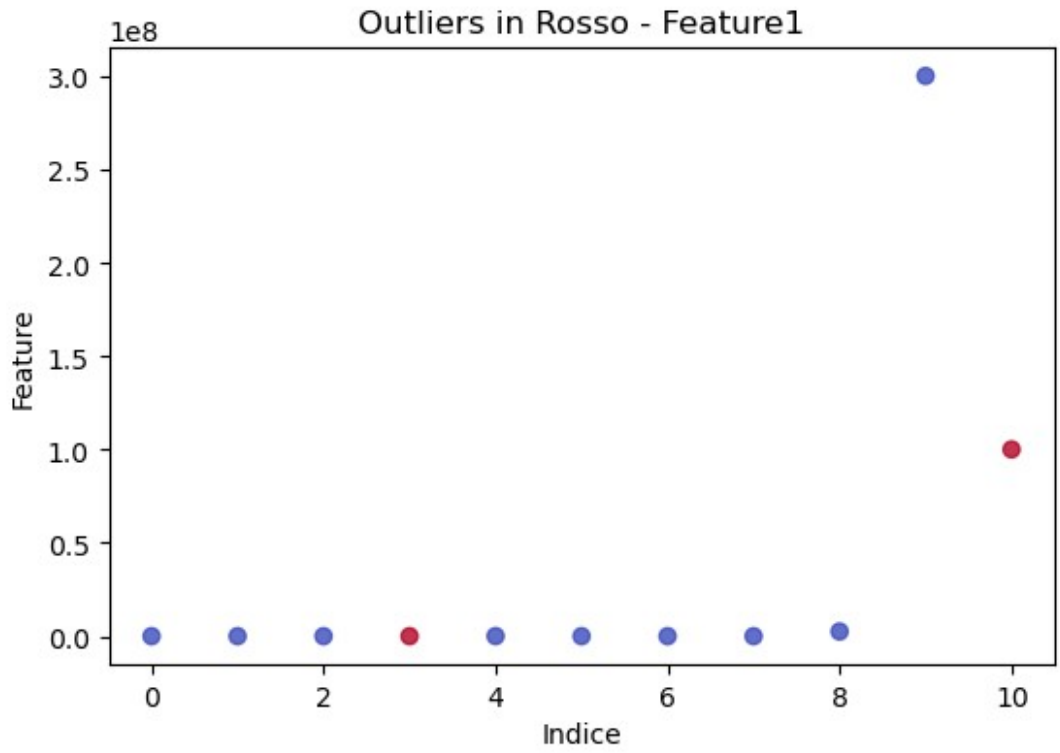
```
df_filtered = df[df['Is_Outlier'] == False ]
df_filtered
```

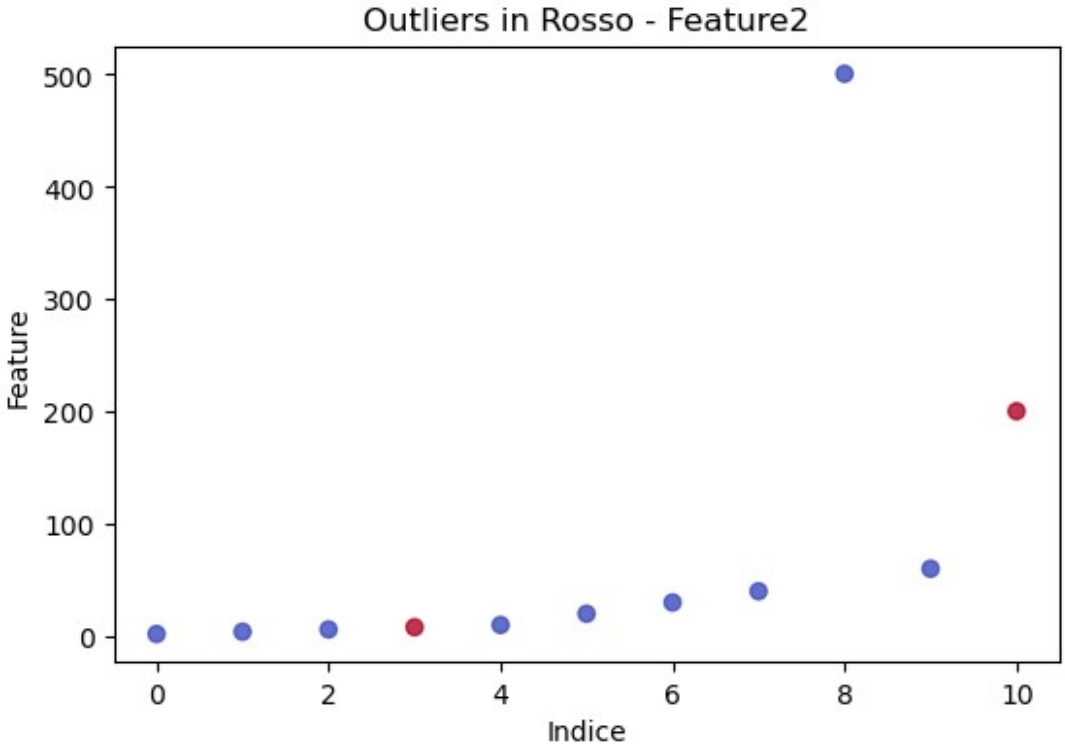
```
    Feature1  Feature2  Feature3  Feature4  Is_Outlier
0          1         2         5         1       False
1        200         4        10   -200000       False
2          3         6        15         3       False
4      50000        10        25         5       False
5         10        20        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
```

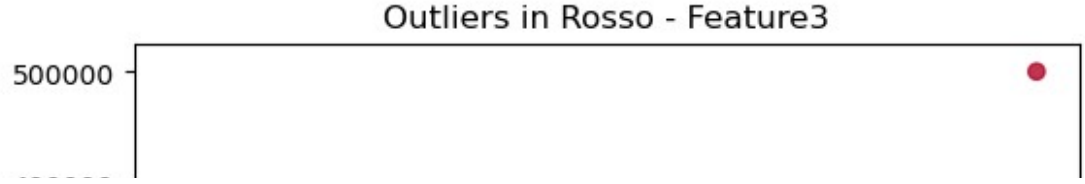
Visualizzazione degli Outlier nelle Feature tramite Matrice di Grafici

```
# 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