


Introduzione a Pandas: Il Potente Strumento di Analisi Dati in Python

Pandas è una libreria Python essenziale che rivoluziona l'analisi e la manipolazione dei dati. Sviluppata specificamente per gestire dati tabellari e sequenziali, questa libreria rappresenta uno strumento fondamentale nel toolkit di ogni data scientist.

In questo corso, esploreremo come Pandas può trasformare il vostro approccio all'analisi dei dati, fornendo esempi pratici e dimostrazioni di sintassi che vi permetteranno di padroneggiare questa potente libreria.

 by **Simone Pipitone**



Gli Oggetti Fondamentali: Series e DataFrame

Series

Una Series è un vettore monodimensionale etichettato che può contenere qualsiasi tipo di dato. È simile a una colonna in Excel o a un array con indici personalizzati.

```
import pandas as pd

# Creazione di una Series
serie = pd.Series([10, 20, 30, 40],
                  index=['a', 'b', 'c', 'd'])
print(serie)
# Output:
# a    10
# b    20
# c    30
# d    40
# dtype: int64
```

DataFrame

Un DataFrame è una struttura bidimensionale simile a una tabella, con righe e colonne etichettate. È l'equivalente di un foglio Excel in Python.

```
# Creazione di un DataFrame
dati = {'Nome': ['Marco', 'Anna', 'Luca'],
        'Età': [28, 34, 29],
        'Città': ['Roma', 'Milano', 'Napoli']}
df = pd.DataFrame(dati)
print(df)
# Output:
#   Nome  Età  Città
# 0 Marco  28   Roma
# 1 Anna  34   Milano
# 2 Luca  29   Napoli
```

Creazione e Caricamento dei Dati

Da Strutture Python

```
# Da liste
df1 = pd.DataFrame([
    [1, 'Paolo', 35],
    [2, 'Maria', 28]],
    columns=['ID', 'Nome', 'Età'])
```

```
# Da dizionari
df2 = pd.DataFrame({
    'Prodotto': ['A', 'B', 'C'],
    'Prezzo': [10.5, 5.0, 20.0]})
```

Da File Esterni

```
# Da CSV
df_csv = pd.read_csv('dati.csv',
    sep=',',
    encoding='utf-8')
```

```
# Da Excel
df_excel = pd.read_excel('dati.xlsx',
    sheet_name='Foglio1')
```

```
# Da JSON
df_json = pd.read_json('dati.json')
```

Pandas offre una flessibilità incredibile nella creazione e nel caricamento dei dati, permettendo di lavorare facilmente con diversi formati e fonti.

Filtrare un DataFrame: Tecniche Essenziali

Metodi di Filtrazione

Filtrare i dati è un'operazione fondamentale nell'analisi. Pandas offre diversi metodi potenti e intuitivi per estrarre esattamente i dati che vi interessano.

Filtro con condizioni booleane

```
maggioirenni = df[df['Età'] > 18]
```

Filtro con multiple condizioni

```
milano_adulti = df[(df['Città'] == 'Milano') &  
  (df['Età'] > 30)]
```

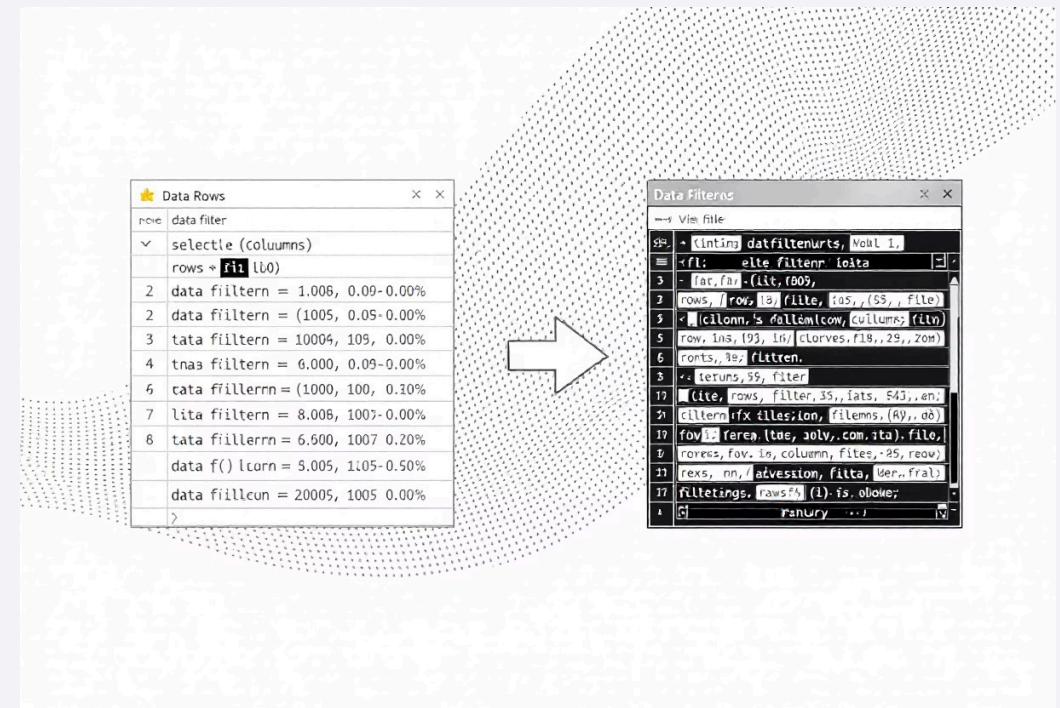
Selezione di colonne specifiche

```
nomi_città = df[['Nome', 'Città']]
```

Utilizzo di .loc e .iloc

```
prima_riga = df.iloc[0] # Per posizione
```

```
dati_marco = df.loc[df['Nome'] == 'Marco'] # Per etichetta
```



Metodi Avanzati

Filtrare valori nulli

```
dati_completi = df.dropna()
```

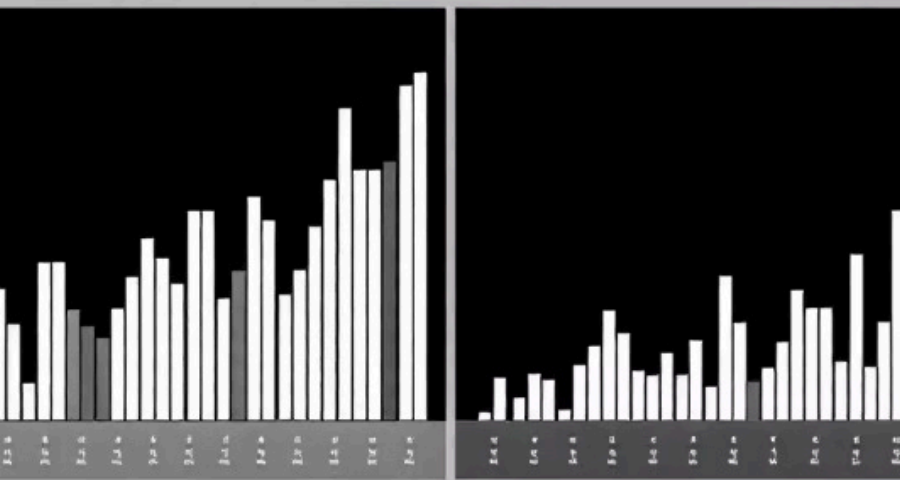
Filtrare con query

```
alti_milano = df.query("Altezza > 175 and Città == 'Milano'")
```

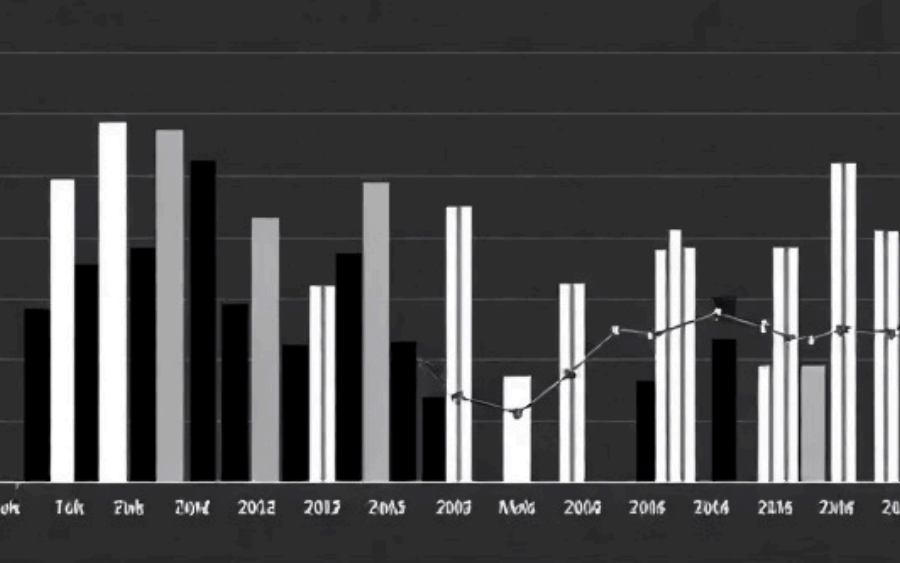
Utilizzo di .isin()

```
città_nord = df[df['Città'].isin(['Milano', 'Torino', 'Venezia'])]
```

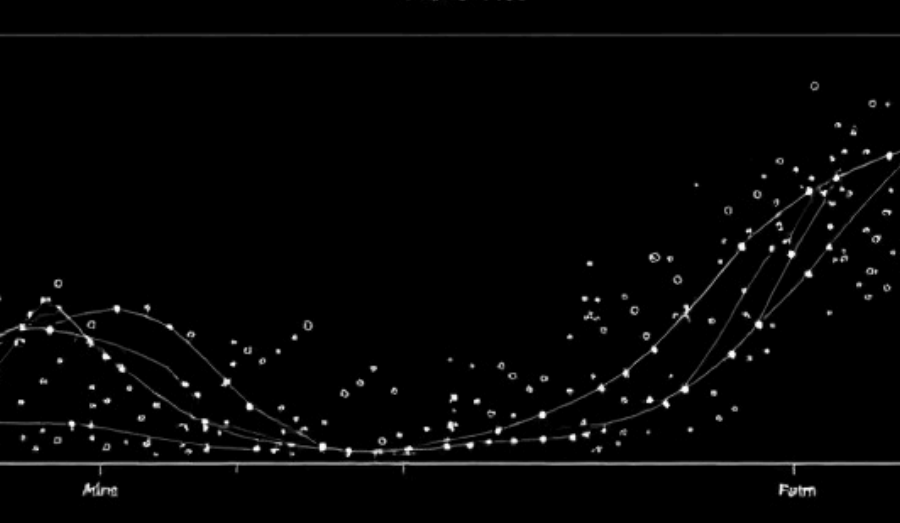

Scatter Plots



Line Plot



Scatter Plot



Visualizzazione Dati con Matplotlib

Matplotlib è la libreria di visualizzazione più utilizzata con Pandas, permettendo di creare grafici professionali dai vostri dati. L'integrazione tra Pandas e Matplotlib è fluida e intuitiva.

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

# Creazione di un DataFrame di esempio
df = pd.DataFrame({
    'Anno': [2018, 2019, 2020, 2021, 2022],
    'Vendite': [120, 145, 132, 170, 190]
})

# Creazione di un grafico lineare
plt.figure(figsize=(10, 6))
plt.plot(df['Anno'], df['Vendite'], marker='o', linestyle='-', color='blue')
plt.title('Andamento Vendite 2018-2022')
plt.xlabel('Anno')
plt.ylabel('Vendite (migliaia €)')
plt.grid(True)
plt.show()

# Metodo diretto di Pandas (usa Matplotlib internamente)
df.plot(x='Anno', y='Vendite', kind='line', figsize=(10, 6), title='Andamento Vendite')
```

La sintassi di base di Matplotlib richiede di specificare i dati degli assi, mentre il metodo `.plot()` di Pandas offre un'interfaccia più semplificata che utilizza internamente Matplotlib.

Esportazione dei Dati in Diversi Formati

Excel

```
# Esportazione in Excel
df.to_excel('dati_esportati.xlsx',
            sheet_name='Risultati',
            index=False) # Rimuove
l'indice

# Esportazione multipli fogli
with pd.ExcelWriter('report.xlsx')
as writer:
    df1.to_excel(writer,
sheet_name='Vendite')
    df2.to_excel(writer,
sheet_name='Clienti')
```

CSV

```
# Esportazione in CSV
df.to_csv('dati_esportati.csv',
          sep=',',
          encoding='utf-8',
          index=False)

# Con parametri avanzati
df.to_csv('report.csv',
          decimal=',', # Separatore
          decimale
          date_format='%d/%m/%Y')
# Formato date
```

JSON e XML

```
# Esportazione in JSON
df.to_json('dati.json',
           orient='records') # Formato record

# Esportazione in XML (richiede pacchetto lxml)
# pip install lxml
df.to_xml('dati.xml',
          root_name='dati',
          row_name='record')
```

L'esportazione dei dati è un passaggio cruciale nel flusso di lavoro dell'analisi dati. Pandas semplifica questa operazione offrendo metodi intuitivi per salvare i dati in formati compatibili con altri software e sistemi.

Esercitazioni Pratiche con Pandas e Matplotlib

Esercizio Base: Analisi Vendite

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

# Caricamento dati
vendite = pd.DataFrame({
    'Prodotto': ['A', 'B', 'C', 'A', 'B', 'C'],
    'Mese': ['Gen', 'Gen', 'Gen', 'Feb', 'Feb', 'Feb'],
    'Quantità': [10, 15, 8, 12, 9, 11]
})

# Analisi: vendite totali per prodotto
totale_prodotti = vendite.groupby('Prodotto')['Quantità'].sum()
print(totale_prodotti)

# Visualizzazione
totale_prodotti.plot(kind='bar', color='skyblue')
plt.title('Vendite Totali per Prodotto')
plt.ylabel('Unità Vendute')
plt.show()
```

Esercizio Avanzato: Analisi Finanziaria

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

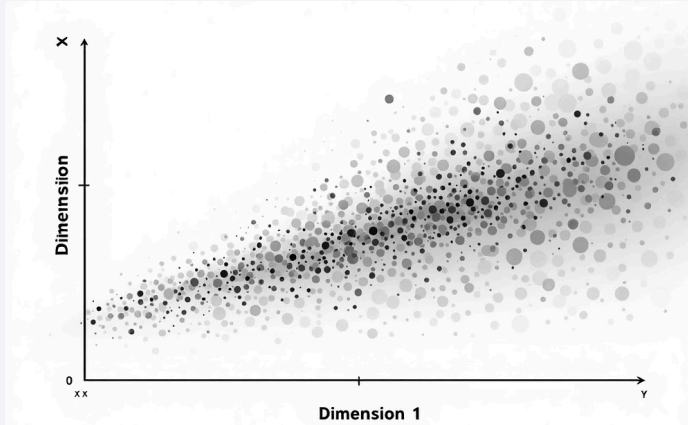
# Simulazione dati finanziari
np.random.seed(42)
date_range = pd.date_range(start='2022-01-01', periods=100,
                             freq='D')
azioni = pd.DataFrame({
    'Data': date_range,
    'Prezzo_A': np.random.normal(100, 5, 100).cumsum() + 100,
    'Prezzo_B': np.random.normal(150, 3, 100).cumsum() + 150
})

# Calcolo rendimento percentuale
azioni['Rendimento_A'] = azioni['Prezzo_A'].pct_change() * 100
azioni['Rendimento_B'] = azioni['Prezzo_B'].pct_change() * 100

# Analisi correlazione
corr = azioni[['Rendimento_A', 'Rendimento_B']].corr()

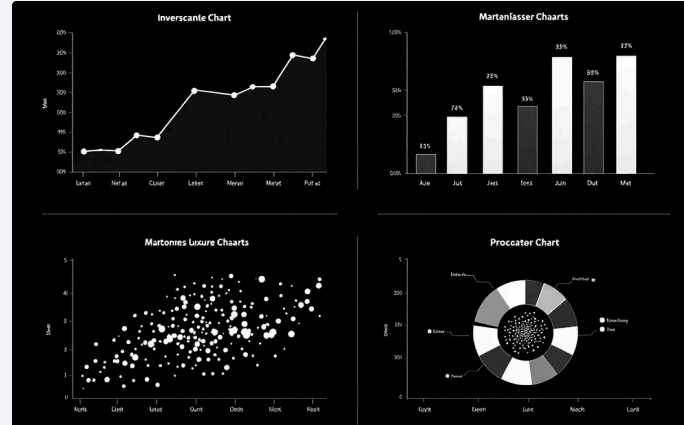
# Visualizzazione andamento prezzi
azioni.plot(x='Data', y=['Prezzo_A', 'Prezzo_B'],
            figsize=(12, 6), title='Andamento Prezzi Azioni')
```

Funzioni Avanzate di Matplotlib



Scatter Plot Avanzati

```
plt.figure(figsize=(10, 6))
plt.scatter(df['x'], df['y'],
            c=df['z'], # Colore basato su valori
            s=df['dim']*50, # Dimensione punti
            alpha=0.7, # Trasparenza
            cmap='viridis') # Mappa colori
plt.colorbar(label='Valore Z')
plt.title('Scatter Plot Avanzato')
```



Subplots

```
# Creazione di una griglia di grafici
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

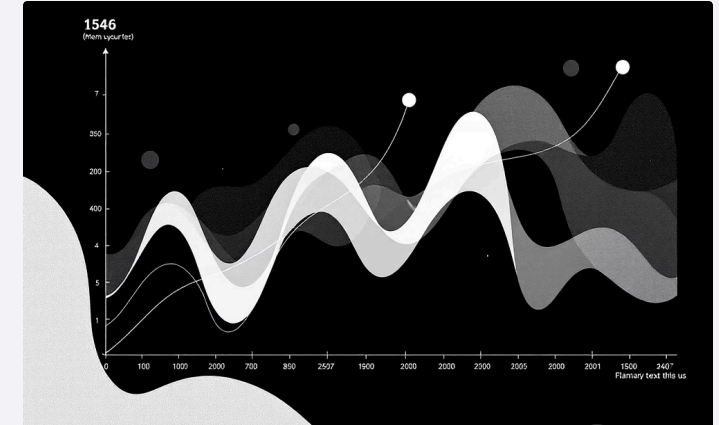
# Primo grafico (riga 0, colonna 0)
axs[0, 0].bar(df['categoria'], df['valori'])
axs[0, 0].set_title('Grafico a Barre')

# Secondo grafico (riga 0, colonna 1)
axs[0, 1].plot(df['x'], df['y'], 'r-')
axs[0, 1].set_title('Grafico Lineare')

# Terzo grafico (riga 1, colonna 0)
axs[1, 0].scatter(df['a'], df['b'])
axs[1, 0].set_title('Scatter Plot')

# Quarto grafico (riga 1, colonna 1)
axs[1, 1].hist(df['dati'], bins=15)
axs[1, 1].set_title('Istogramma')

plt.tight_layout() # Ottimizza spaziatura
```



Personalizzazione Avanzata

```
plt.figure(figsize=(12, 8))
plt.plot(df['tempo'], df['temperatura'], 'b-',
         linewidth=2)

# Aggiunta di annotazioni
plt.annotate('Picco',
            xy=(df['tempo'][10], df['temperatura'][10]),
            xytext=(df['tempo'][10]+2,
                    df['temperatura'][10]+5),
            arrowprops=dict(facecolor='black',
                             shrink=0.05))

# Personalizzazione stile
plt.grid(True, linestyle='--', alpha=0.7)
plt.axhline(y=25, color='r', linestyle='-',
            alpha=0.3)
plt.fill_between(df['tempo'],
                df['temperatura'],
                alpha=0.2, color='skyblue')
```


Funzioni Avanzate di Pandas

Aggregazione e Pivot

```
# GroupBy avanzato con multiple aggregazioni
risultato = df.groupby('Categoria').agg({
    'Vendite': ['sum', 'mean', 'std'],
    'Clienti': ['count', 'max'],
    'Profitto': ['min', 'max', 'mean']
})

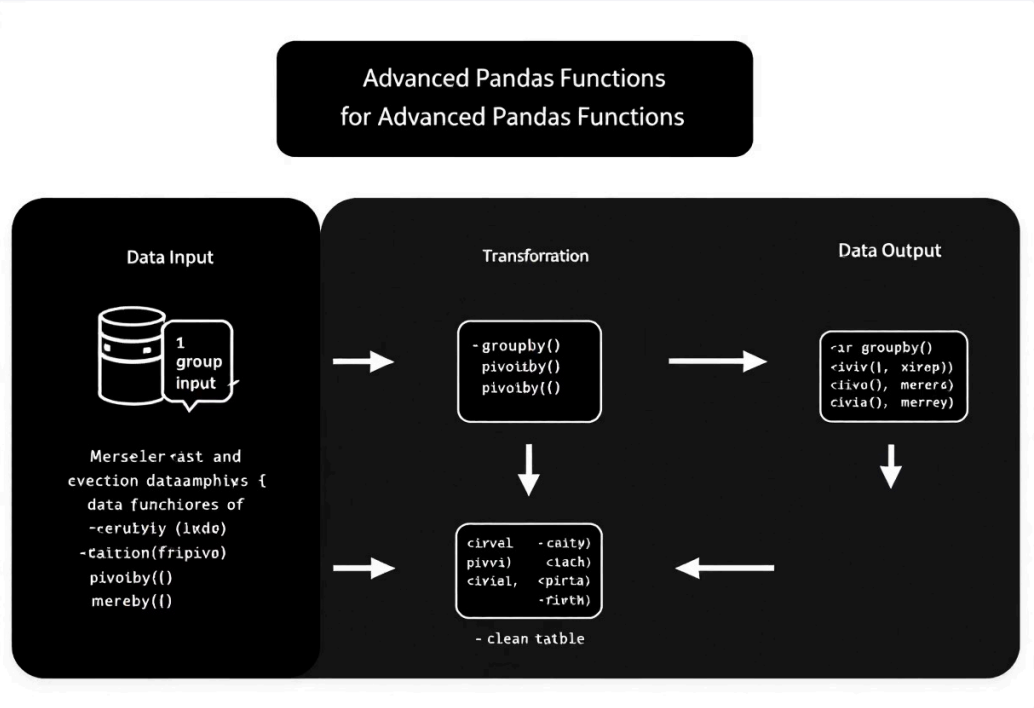
# Pivot Table
tabella_pivot = pd.pivot_table(
    df,
    values='Vendite',
    index=['Regione', 'Città'],
    columns='Prodotto',
    aggfunc='sum',
    fill_value=0,
    margins=True # Aggiunge totali
)
```

Manipolazione Avanzata

```
# Applicazione di funzioni personalizzate
def calcola_categoria(valore):
    if valore < 100: return 'Basso'
    elif valore < 500: return 'Medio'
    else: return 'Alto'

df['Categoria'] = df['Valore'].apply(calcola_categoria)

# Operazioni sulle finestre temporali
df_finestra = df.set_index('Data')
medie_mobili = df_finestra['Valore'].rolling(window=7).mean()
```



Gestione Dati Mancanti

```
# Identificazione valori mancanti
mancanti = df.isna().sum()

# Imputazione avanzata
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df_imputato = pd.DataFrame(
    imputer.fit_transform(df),
    columns=df.columns
)

# Interpolazione per serie temporali
df['Temperatura'] = df['Temperatura'].interpolate(
    method='cubic'
)
```

Unione Dataset

```
# Merge avanzato
df_completo = pd.merge(
    clienti,
    ordini,
    how='left', # tipo di join
    left_on=['ID_Cliente', 'Regione'],
    right_on=['Cliente_ID', 'Area'],
    suffixes=('_clienti', '_ordini')
)
```

Introduzione a Seaborn: Visualizzazione Statistica Avanzata

Cos'è Seaborn?

Seaborn è una libreria di visualizzazione statistica basata su Matplotlib che offre un'interfaccia di alto livello per creare grafici statistici informativi ed esteticamente gradevoli.

Vantaggi di Seaborn

- Temi predefiniti professionali
- Integrazione perfetta con Pandas
- Visualizzazioni statistiche specializzate
- Grafici multivariati semplificati
- Gestione automatica dei dati categorici

Esempio Base

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

# Configurazione stile
sns.set_theme(style="whitegrid")

# Caricamento dataset di esempio
df = sns.load_dataset("tips")

# Creazione di un grafico base
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df,
    x="total_bill",
    y="tip",
    hue="time",
    size="size",
    palette="deep"
)
plt.title("Relazione tra Conto Totale e Mancia")
plt.show()
```

Da Matplotlib a Seaborn

Seaborn estende le funzionalità di Matplotlib, rendendo più semplice creare visualizzazioni statistiche complesse con poche righe di codice, mantenendo la flessibilità di personalizzazione di Matplotlib.