



Specifiche pandas

MATERIA

Introduzione a Pandas

Pandas è costruito sopra NumPy, un'altra libreria Python che fornisce supporto per array multidimensionali, e integra bene con altre librerie come Matplotlib per la visualizzazione dei dati e scikit-learn per il machine learning. Pandas introduce principalmente due nuove strutture di dati:

Series e DataFrame.

- **Series:** Una Serie è un array unidimensionale etichettato capace di contenere dati di qualsiasi tipo (interi, stringhe, numeri a virgola mobile, oggetti Python, etc.). L'asse delle etichette di una Serie viene chiamato indice.
- **DataFrame:** Un DataFrame è una struttura dati bidimensionale, con dati allineati in un formato tabellare. È simile a un foglio di calcolo o a una tabella SQL. Un DataFrame supporta colonne di diversi tipi e può essere visto come un dizionario di Serie, dove ogni colonna è una Serie.



Componenti Principali di Pandas

- **Gestione dei dati:** Pandas fornisce funzionalità avanzate per il caricamento, la pulizia, l'esplorazione e la manipolazione dei dati. Ciò include funzioni per gestire valori mancanti, unire e concatenare dataset, trasformare e aggregare dati, e molto altro.
- **Indicizzazione e selezione:** La libreria offre strumenti potenti per l'indicizzazione e la selezione dei dati, permettendo di accedere a sottinsiemi di dati in modo efficiente. Ciò include l'uso di etichette (loc), posizioni numeriche (iloc), e una combinazione di condizioni booleane.



- **Gestione del tempo:** Pandas ha un forte supporto per le serie temporali, consentendo di lavorare con date e orari in vari formati, gestire serie temporali con frequenze diverse, e eseguire operazioni di time-series come lo shifting e il resampling.
- **Input/Output:** Pandas supporta la lettura e la scrittura di dati da e verso molti formati, come CSV, Excel, SQL, JSON, HTML, e HDF5, facilitando l'interazione con diverse fonti di dati.
- **Visualizzazione:** Sebbene Pandas non sia una libreria di visualizzazione, integra bene con Matplotlib e Seaborn per creare grafici direttamente dai DataFrame, rendendo l'esplorazione dei dati visuale e interattiva.



Analisi e Prestazioni

Pandas è ottimizzato per prestazioni in molte operazioni di data science, grazie all'uso sottostante di NumPy per operazioni su array.

Per dataset molto grandi, Pandas può essere utilizzato insieme a librerie come Dask o Vaex per calcoli paralleli o su grandi dataset che non entrano nella memoria RAM.



- **Gestione dei Branch:** Offre strumenti visivi per la creazione, il cambio, e la fusione di branch, rendendo più semplice la gestione delle varie linee di sviluppo.
- **Storico e Differenze:** Visualizza lo storico dei commit e le differenze tra le versioni del codice, rendendo facile tracciare e comprendere le modifiche



Pandas offre una vasta gamma di metodi che lo rendono uno strumento potente per l'analisi dei dati. Ecco un'analisi di alcuni dei metodi più utilizzati in Pandas, focalizzandosi su come vengono applicati nelle operazioni comuni di manipolazione dei dati.

Metodi di Lettura e Scrittura

- `pd.read_csv()`: Carica dati da un file CSV in un DataFrame. È uno dei metodi di input più comuni, grazie alla popolarità del formato CSV per lo scambio di dati.
- `pd.to_csv()`: Esporta un DataFrame in un file CSV, permettendo di salvare i risultati delle analisi in un formato facilmente condivisibile e leggibile.



Esplorazione e Pulizia dei Dati

- `df.head()`: Visualizza le prime N righe di un DataFrame (5 per default), utile per un primo esame dei dati.
- `df.describe()`: Fornisce un riepilogo statistico delle colonne numeriche, utile per avere una visione generale delle distribuzioni e identificare eventuali valori anomali.
- `df.dropna()`: Rimuove le righe con valori mancanti, essenziale nella fase di pulizia dei dati.
- `df.fillna()`: Sostituisce i valori mancanti con un valore specificato, permettendo di mantenere dati completi anche in presenza di lacune.



Manipolazione dei Dati

- **df.loc[]**: Seleziona dati in base alle etichette delle righe e delle colonne. È molto flessibile, permettendo selezioni condizionali e accesso a sottogruppi di dati.
- **df.iloc[]**: Seleziona dati in base alla posizione numerica delle righe e delle colonne. È utile quando l'ordine dei dati è importante o quando si lavora con indici numerici.
- **df.groupby()**: Raggruppa i dati in base a una o più colonne e consente di applicare funzioni di aggregazione (come somma, media, etc.) sui gruppi, essenziale per l'analisi aggregata dei dati.
- **df.merge()**: Combina due DataFrame in base a una o più chiavi comuni, simile all'operazione di JOIN in SQL. È cruciale per unire dati da fonti diverse.



Modifica della Struttura dei Dati

- **df.pivot_table():** Crea una tabella pivot da un DataFrame, organizzando i dati in un formato che facilita l'analisi comparativa.
- **df.melt():** Trasforma un DataFrame da un formato wide a un formato long, utile per ripristinare un formato adatto alle analisi dopo l'uso di pivot_table().



Serie Temporali

- **pd.to_datetime():** Converte una colonna in formato datetime, indispensabile per lavorare con serie temporali.
- **df.resample():** Riaggrega una serie temporale su una frequenza specifica, utile per analisi periodiche come dati giornalieri, mensili, o annuali.



Serie Temporali

- **pd.to_datetime():** Converte una colonna in formato datetime, indispensabile per lavorare con serie temporali.
- **df.resample():** Riaggrega una serie temporale su una frequenza specifica, utile per analisi periodiche come dati giornalieri, mensili, o annuali.



Per dimostrare le capacità avanzate di Pandas, creiamo un esempio che attraversa diverse fasi dell'analisi dei dati: il caricamento, la pulizia, la manipolazione e l'aggregazione, fino alla visualizzazione.

Immagina di avere un dataset di vendite di un negozio, contenente informazioni su prodotti, quantità vendute, prezzi e date di vendita.

Questo esempio assumerà che hai un file CSV vendite.csv con le seguenti colonne: Data, Prodotto, Quantità, PrezzoUnitario.



```
1.import pandas as pd
2.import matplotlib.pyplot as plt
3.
4.# Caricamento dei dati
5.df = pd.read_csv('vendite.csv', parse_dates=['Data'])
6.
7.# Pulizia dei dati
8.# Rimuoviamo eventuali righe con valori mancanti
9.df.dropna(inplace=True)
10.
11.# Assicuriamoci che Quantità e PrezzoUnitario siano positivi
12.df = df[(df['Quantità'] > 0) & (df['PrezzoUnitario'] > 0)]
13.
14.# Aggiungiamo una colonna per il totale vendite
15.df['TotaleVendite'] = df['Quantità'] * df['PrezzoUnitario']
16.
17.# Manipolazione dei dati
18.# Aggiungiamo colonne per anno e mese per analisi temporale
19.df['Anno'] = df['Data'].dt.year
20.df['Mese'] = df['Data'].dt.month
21.
22.# Aggregazione dei dati
23.# Calcoliamo le vendite totali per prodotto
24.vendite_per_prodotto = df.groupby('Prodotto')['TotaleVendite'].sum().sort_values(ascending=False)
25.
26.# Calcoliamo le vendite mensili totali
27.vendite_mensili = df.groupby(['Anno', 'Mese'])['TotaleVendite'].sum()
28.
29.# Visualizzazione
30.# Grafico delle vendite totali per prodotto
31.vendite_per_prodotto.head(10).plot(kind='bar', title='Top 10 Prodotti per Vendite Totali')
32.plt.xlabel('Prodotto')
33.plt.ylabel('Totale Vendite')
34.plt.show()
35.
36.# Grafico delle vendite mensili
37.vendite_mensili.plot(kind='line', title='Vendite Mensili')
38.plt.xlabel('Mese')
39.plt.ylabel('Totale Vendite')
40.plt.show()
```



Spiegazione del Codice

1. **Caricamento dei dati:** Carichiamo il dataset da un file CSV, specificando che la colonna Data deve essere interpretata come una data.
2. **Pulizia dei dati:** Rimuoviamo le righe con valori mancanti e filtriamo i dati per assicurarci che le quantità e i prezzi siano positivi, rimuovendo eventuali dati errati o non validi.
3. **Manipolazione dei dati:** Aggiungiamo una colonna TotaleVendite per rappresentare il totale delle vendite per ogni transazione. Introduciamo anche colonne per l'Anno e il Mese estratti dalla data di vendita, per facilitare analisi temporali.
4. **Aggregazione dei dati:** Raggruppiamo i dati per prodotto e per coppia anno-mese, calcolando il totale delle vendite. Questo ci permette di analizzare quali prodotti generano maggiori entrate e come le vendite variano nel tempo.
5. **Visualizzazione:** Creiamo due grafici: uno a barre per le vendite totali dei top 10 prodotti e uno lineare per le vendite mensili nel tempo. Questi grafici offrono una visione immediata delle performance di vendita.



```
1.import pandas as pd
2.import matplotlib.pyplot as plt
3.
4.# Caricamento dei dati
5.vendite = pd.read_csv('vendite_giornaliere.csv', parse_dates=['Data'])
6.prodotti = pd.read_csv('info_prodotti.csv')
7.
8.# Unione dei dataset
9.df = pd.merge(vendite, prodotti, on='IDProdotto')
10.
11.# Calcolo della media mobile a 7 giorni per prodotto
12.df.sort_values(['NomeProdotto', 'Data'], inplace=True) # Assicurarsi che i dati siano ordinati
13.df['MediaMobile7gg'] = df.groupby('NomeProdotto')['Quantità'].transform(lambda x: x.rolling(window=7).mean())
14.
15.# Raggruppamento delle vendite per categoria
16.vendite_per_categoria = df.groupby('Categoria')['Quantità'].sum()
17.
18.# Reshaping dei dati per analisi temporale per categoria
19.vendite_temporali = df.pivot_table(index='Data', columns='Categoria', values='Quantità', aggfunc='sum').fillna(0)
20.
21.# Visualizzazione delle vendite totali per categoria
22.vendite_per_categoria.plot(kind='bar', title='Vendite Totali per Categoria')
23.plt.xlabel('Categoria')
24.plt.ylabel('Quantità Venduta')
25.plt.show()
26.
27.# Visualizzazione delle vendite giornaliere per categoria
28.vendite_temporali.plot(title='Vendite Giornaliere per Categoria', figsize=(10, 6))
29.plt.xlabel('Data')
30.plt.ylabel('Quantità Venduta')
31.plt.legend(title='Categoria')
32.plt.show()
```



Spiegazione del Codice

1. **Unione dei dataset:** Combiniamo le informazioni sulle vendite con quelle sui prodotti usando `pd.merge()`, unendoli per `IDProdotto`. Questo arricchisce il dataset di vendite con dettagli sui prodotti venduti.
2. **Media mobile delle vendite:** Utilizziamo `groupby()` e `transform()` per calcolare una media mobile a 7 giorni delle vendite per ciascun prodotto. Questo aiuta a identificare tendenze a breve termine nelle vendite, levigando le fluttuazioni giornaliere.
3. **Raggruppamento per categoria:** Aggregiamo le vendite per categoria di prodotto, fornendo una visione del volume di vendite per categoria.
4. **Reshaping per analisi temporale:** Con `pivot_table()`, trasformiamo i dati per avere le date sull'asse delle righe e le categorie sui titoli delle colonne, facilitando l'analisi delle tendenze di vendita per categoria nel tempo.
5. **Visualizzazione:** Creiamo un grafico a barre per mostrare le vendite totali per categoria e un grafico lineare per esaminare come le vendite giornaliere variano per categoria nel tempo.



Tecniche Avanzate con Pandas

Analisi delle Serie Temporal

Pandas offre strumenti potenti per lavorare con serie temporali, rendendolo ideale per analizzare dati temporali complessi. Le funzionalità includono:

- **Resampling e Windowing:** Permette di aggregare o trasformare dati temporali su intervalli regolari o applicare funzioni su finestre di tempo, come medie mobili o esponenziali.
- **Time Zone Handling:** Gestisce conversioni tra fusi orari e operazioni consapevoli del fuso orario.
- **Periodi e Frequenze:** Supporta periodi di tempo specifici e la loro conversione, facilitando l'analisi su basi temporali fisse come mensile o annuale.



Tecniche Avanzate con Pandas

Operazioni Avanzate di Manipolazione dei Dati

Oltre alle funzionalità di base, Pandas include metodi avanzati per manipolazioni complesse dei dati, come:

- **Categorical Data:** Supporta dati categorici, che possono aumentare le performance e ridurre l'uso di memoria, specialmente su grandi dataset.
- **Query Method:** Permette di interrogare i DataFrame con espressioni booleane, semplificando la selezione di sottogruppi di dati.
- **MultiIndex:** Gestisce DataFrame con più livelli di indici (indice gerarchico), utile per rappresentare dati ad alta dimensionalità in forma tabulare.



In conclusione, la padronanza di Pandas apre la porta a un'ampia gamma di analisi dati e applicazioni, dalle operazioni più semplici a tecniche di analisi avanzate.

La sua integrazione con altre librerie di Python per la visualizzazione e l'analisi statistica ne fa un pilastro fondamentale nell'ecosistema dei dati di Python.



Buon MasterD a tutti

