



Fondamenti Pandas prt 2

Python

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.



Caricamento e visualizzazione dei dati

Contesto: Supponiamo di avere un file CSV contenente dati sui voti degli studenti e vogliamo caricare questi dati in un DataFrame per analizzarli.

```
1.import pandas as pd  
2.  
3.# Caricamento dei dati da un file CSV  
4.df = pd.read_csv('dati_studenti.csv')  
5.  
6.# Visualizzazione delle prime 5 righe del DataFrame  
7.print(df.head())
```

Spiegazione:

- Importiamo pandas e lo rinominiamo come pd per brevità.
- Carichiamo i dati da un file CSV usando il metodo read_csv.
- Usiamo il metodo head() per ottenere e visualizzare le prime cinque righe del DataFrame.



Filtraggio dei dati

Contesto: Vogliamo filtrare gli studenti che hanno ottenuto un voto superiore a 7.

1. # Filtraggio degli studenti con voto superiore a 7
2. studenti_promossi = df[df['Voto'] > 7]
- 3.
4. # Visualizzazione del risultato
5. print(studenti_promossi)

Spiegazione:

- Usiamo un'operazione di filtraggio condizionale (`df['Voto'] > 7`) per selezionare solo le righe dove il voto è superiore a 7.
- Il risultato è un nuovo DataFrame contenente solo gli studenti che soddisfano la condizione.



Aggregazione e statistiche

Contesto: Calcoliamo la media dei voti degli studenti.

```
1.# Calcolo della media dei voti  
2.media_voti = df['Voto'].mean()  
3.  
4.# Stampa della media  
5.print(f"La media dei voti è: {media_voti:.2f}")
```

Spiegazione:

- Utilizziamo il metodo `mean()` sulla colonna `Voto` per calcolare la media.
- Stampiamo la media arrotondata a due cifre decimali.



Unione di DataFrame

Contesto: Supponiamo di avere due set di dati, uno con i dettagli degli studenti (nome, ID) e un altro con i voti degli studenti per diversi esami. Vogliamo unire questi due DataFrame per avere un unico DataFrame che contenga tutte le informazioni.

```
1.import pandas as pd
2.
3.# Creazione di due DataFrame
4.df_studenti = pd.DataFrame({
5.    'ID': [1, 2, 3],
6.    'Nome': ['Alice', 'Bob', 'Charlie']
7.})
8.
9.df_voti = pd.DataFrame({
10.   'ID': [1, 2, 3],
11.   'Matematica': [6, 7, 8],
12.   'Fisica': [5, 9, 7]
13.})
14.
15.# Unione dei DataFrame su 'ID'
16.df_completo = pd.merge(df_studenti, df_voti, on='ID')
17.
18.# Visualizzazione del DataFrame risultante
19.print(df_completo)
20.
```



Spiegazione:

- Creiamo due DataFrame separati per studenti e voti.
- Usiamo pd.merge() per unire i DataFrame basandoci sulla colonna 'ID'.
- Il risultato è un DataFrame che combina le informazioni di entrambi i set di dati.

Gestione delle date

Contesto: Abbiamo un DataFrame che contiene le date degli esami e i voti corrispondenti. Vogliamo aggiungere una colonna che indichi il giorno della settimana in cui è stato sostenuto l'esame.

```
1.# DataFrame con date e voti  
2.df_esami = pd.DataFrame({  
3.    'Data': pd.to_datetime(['2023-09-01', '2023-09-02', '2023-09-03']),  
4.    'Voto': [24, 30, 18]  
5.})  
6.  
7.# Aggiunta della colonna 'Giorno della settimana'  
8.df_esami['Giorno'] = df_esami['Data'].dt.day_name()  
9.  
10.# Visualizzazione del DataFrame aggiornato  
11.print(df_esami)  
12.
```

Spiegazione:

- Convertiamo le stringhe di date in oggetti datetime usando pd.to_datetime().
- Aggiungiamo una colonna Giorno che usa il metodo dt.day_name() per ottenere il nome del giorno dalla data.



Operazioni di gruppo e aggregazione

Contesto: Vogliamo calcolare il voto medio in ciascuna materia e il numero di studenti che hanno sostenuto ogni esame.

```
1.import pandas as pd
2.
3.# DataFrame con materie e voti
4.df_materie = pd.DataFrame({
5.    'Materia': ['Matematica', 'Fisica', 'Matematica', 'Fisica', 'Matematica'],
6.    'Voto': [20, 25, 22, 20, 30]
7.})
8.
9.# Calcolo delle statistiche per materia
10.statistiche = df_materie.groupby('Materia').agg({
11.    'Voto': ['mean', 'count']
12.})
13.
14.# Rinomina delle colonne per chiarezza
15.statistiche.columns = ['Media Voti', 'Numero Studenti']
16.
17.# Visualizzazione dei risultati
18.print(statistiche)
19.
```



Spiegazione:

- Usiamo groupby() per raggruppare i dati per materia.
- agg() permette di applicare funzioni di aggregazione specifiche, in questo caso la media e il conteggio.
- Rinominiamo le colonne per rendere il DataFrame più chiaro.

Pandas offre una vasta gamma di tecniche avanzate per l'analisi dei dati.

Di seguito, discuteremo alcune di queste tecniche, includendo il reshaping dei dati, l'utilizzo di operazioni di windowing, e la manipolazione di dati testuali.

Per ogni tecnica, fornirò un contesto, un esempio di codice dettagliatamente commentato, e una spiegazione del processo.



Reshaping di dati con pivot_table

Contesto: Supponiamo di avere un dataset che registra i voti degli studenti per diverse materie in più sessioni d'esame. Desideriamo riorganizzare il DataFrame per avere una visione aggregata che mostri la media dei voti per ogni materia per sessione.

```
1. import pandas as pd  
2.  
3. # Creazione di un DataFrame di esempio  
4. df = pd.DataFrame({  
5.     'Sessione': ['2023-01', '2023-01', '2023-02', '2023-02', '2023-01'],  
6.     'Materia': ['Matematica', 'Fisica', 'Matematica', 'Fisica', 'Chimica'],  
7.     'Voto': [24, 18, 30, 25, 22]  
8. })  
9.  
10. # Utilizzo di pivot_table per riorganizzare i dati  
11. pivot = df.pivot_table(index='Sessione', columns='Materia', values='Voto', aggfunc='mean')  
12.  
13. # Visualizzazione del pivot table  
14. print(pivot)
```

Spiegazione:

- Creiamo un DataFrame con sessioni, materie e voti.
- Usiamo pivot_table per trasformare i dati. L'indice index rappresenta le righe (le sessioni), le columns le colonne (le materie), e values i valori che vogliamo analizzare (i voti).
- Specifichiamo aggfunc='mean' per calcolare la media dei voti per materia per ogni sessione.
- Il risultato è una tabella che facilita la comparazione dei voti medi per materia e per sessione.



Finestre scorrevoli (rolling windows)

Contesto: Vogliamo analizzare le tendenze dei voti di matematica nel tempo, calcolando una media mobile per suavizzare le fluttuazioni a breve termine nei dati.

```
1.# Aggiunta di una colonna 'Data' per rappresentare il tempo  
2.df['Data'] = pd.to_datetime(['2023-01-15', '2023-01-31', '2023-02-15', '2023-02-28', '2023-01-20'])  
3.  
4.# Ordinamento del DataFrame per data  
5.df = df.sort_values('Data')  
6.  
7.# Calcolo della media mobile su una finestra di 2 osservazioni  
8.df['Media_mobile'] = df[df['Materia'] == 'Matematica']['Voto'].rolling(window=2).mean()  
9.  
10.# Visualizzazione dei risultati  
11.print(df[['Data', 'Materia', 'Voto', 'Media_mobile']])
```

Spiegazione:

- Aggiungiamo e convertiamo una colonna 'Data' per organizzare i voti nel tempo.
- Ordiniamo i dati per data per assicurare che la media mobile sia calcolata in sequenza temporale.
- Applichiamo `rolling(window=2).mean()` per calcolare la media mobile di due punti dati consecutivi.
- Il DataFrame risultante mostra i voti originali e la media mobile, aiutando a identificare le tendenze.



Manipolazione di dati testuali

Contesto: Abbiamo un dataset che include nomi di studenti in formati non uniformi e vogliamo standardizzare questi nomi per facilitare l'analisi.

```
1.# DataFrame con nomi non uniformi  
2.df_nomi = pd.DataFrame({  
3.    'Nome': ['alice', 'Bob', 'CHARLIE', 'Deborah']  
4.})  
5.  
6.# Standardizzazione dei nomi: prima lettera maiuscola, il resto minuscolo  
7.df_nomi['Nome'] = df_nomi['Nome'].str.capitalize()  
8.  
9.# Visualizzazione dei nomi standardizzati  
10.print(df_nomi)  
11.
```

Spiegazione:

- Il DataFrame originale contiene nomi con maiuscole e minuscole miste.
- Usiamo str.capitalize() per uniformare il formato dei nomi: la prima lettera in maiuscolo e il resto in minuscolo.
- Il risultato è una lista di nomi omogenea, più facile da confrontare e analizzare.



Manipolazione di dati testuali

Contesto: Abbiamo un dataset che include nomi di studenti in formati non uniformi e vogliamo standardizzare questi nomi per facilitare l'analisi.

```
1.# DataFrame con nomi non uniformi  
2.df_nomi = pd.DataFrame({  
3.    'Nome': ['alice', 'Bob', 'CHARLIE', 'Deborah']  
4.})  
5.  
6.# Standardizzazione dei nomi: prima lettera maiuscola, il resto minuscolo  
7.df_nomi['Nome'] = df_nomi['Nome'].str.capitalize()  
8.  
9.# Visualizzazione dei nomi standardizzati  
10.print(df_nomi)  
11.
```

Spiegazione:

- Il DataFrame originale contiene nomi con maiuscole e minuscole miste.
- Usiamo str.capitalize() per uniformare il formato dei nomi: la prima lettera in maiuscolo e il resto in minuscolo.
- Il risultato è una lista di nomi omogenea, più facile da confrontare e analizzare.



Creazione e Caricamento

- `pd.read_csv()`: Carica dati da un file CSV in un DataFrame.
- `pd.read_excel()`: Carica dati da un file Excel in un DataFrame.
- `pd.DataFrame()`: Crea un DataFrame da vari formati di dati come dizionari, liste di liste, o altri DataFrame.
- `pd.Series()`: Crea una serie di dati.

Esplorazione e Pulizia

- `df.head()`: Mostra le prime N righe del DataFrame.
- `df.tail()`: Mostra le ultime N righe del DataFrame.
- `df.describe()`: Mostra statistiche descrittive di base come media, mediana, deviazione standard, ecc.
- `df.info()`: Fornisce una sintesi del DataFrame, incluso il numero di non-null in ogni colonna.
- `df.dropna()`: Rimuove righe con valori mancanti.
- `df.fillna()`: Sostituisce i valori mancanti con un valore specificato.
- `df.drop_duplicates()`: Rimuove le righe duplicate.

Selezione e Filtraggio

- `df.loc[]`: Accesso basato su etichette.
- `df.iloc[]`: Accesso basato su posizione.
- `df[condition]`: Filtraggio dei dati basato su una condizione.
- `df.isna()`: Rileva valori mancanti.

Manipolazione dei Dati

- `df.sort_values()`: Ordina il DataFrame basato su uno o più campi.
- `df.groupby()`: Raggruppa i dati per uno o più attributi e permette l'applicazione di funzioni di aggregazione.
- `df.pivot()`: Reshapes data (produce una "pivot table") basata su valori di colonna.
- `df.pivot_table()`: Crea una tabella pivot come un foglio di calcolo Excel, con valori aggregati.
- `df.merge()`: Effettua operazioni di tipo SQL join.
- `df.concat()`: Concatena DataFrame lungo un asse specifico.

Trasformazioni

- `df.apply()`: Applica una funzione lungo un asse del DataFrame.
- `df.map()`: Applica una funzione elemento per elemento su una Serie.
- `df.applymap()`: Applica una funzione elemento per elemento su un DataFrame.

Output

- `df.to_csv()`: Scrive il DataFrame in un file CSV.
- `df.to_excel()`: Scrive il DataFrame in un file Excel.
- `df.to_sql()`: Scrive il DataFrame in un database SQL.
- `df.to_json()`: Serializza il DataFrame in una stringa JSON.

Time Series

- `pd.to_datetime()`: Converte un argomento in datetime.
- `df.resample()`: Converte la frequenza dei dati temporali.
- `df.asfreq()`: Cambia la frequenza dei dati temporali.



Buon MasterD a tutti

