lez_08

January 21, 2026

1 Pandas

Pandas è una potente libreria di manipolazione e analisi dei dati per Python. Fornisce strutture di dati come serie e dataframe per pulire, trasformare e analizzare in modo semplice ed efficace grandi set di dati e si integra perfettamente con altre librerie Python, come numPy e matplotlib. Offre potenti funzioni per la trasformazione, l'aggregazione e la visualizzazione dei dati, fondamentali per un'analisi efficace.

Pandas ruota attorno a due strutture di dati primarie: serie (1D) per singole colonne e dataframe (2D) per dati tabulari, consentendo una manipolazione efficiente dei dati.

1.1 DataFrame

Un DataFrame è una struttura dati bidimensionale simile a una tabella con righe e colonne etichettate, in cui ogni colonna può avere un tipo di dati diverso (ad esempio, numeri interi, stringhe, numeri in virgola mobile). Può essere creato da strutture dati Python come liste o dizionari:

[1]:
```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
```

In questo esempio, viene creato un dizionario denominato *data* con chiavi che rappresentano i nomi delle colonne (Nome, Età) e valori sotto forma di liste contenenti i rispettivi dati. La funzione *pd.DataFrame()* viene quindi utilizzata per convertire questo dizionario in un **DataFrame**, che viene memorizzato nella variabile df.

1.2 Series

Una serie in Pandas è un array unidimensionale etichettato in grado di contenere qualsiasi tipo di dati (interi, stringhe, valori float, ecc.). Ogni elemento è associato a un indice, predefinito (0, 1, 2…) o personalizzato. Può essere creata da liste, array NumPy, dizionari o valori scalari:

[2]:
```
age_series = pd.Series([25, 30, 35], index=['Alice', 'Bob', 'Charlie'])
print(age_series)
```

```
Alice      25
Bob        30
Charlie    35
dtype: int64
```

1.3 Importare un file CSV

I file CSV (Comma Separated Values) sono un formato comune per archiviare grandi set di dati in testo semplice. La libreria Pandas in Python fornisce la funzione read_csv() per caricare questi file in un DataFrame. Per il nostro esempio useremo *people.csv*

[4]:
```
df = pd.read_csv("./data/people_data.csv")
print(df)
```

```
  First Name Last Name     Sex                         Email Date of birth  \
0     Shelby   Terrell    Male         elijah57@example.net    1945-10-26
1    Phillip   Summers  Female        bethany14@example.com    1910-03-24
2   Kristine    Travis    Male        bthompson@example.com    1992-07-02
3    Yesenia  Martinez    Male    kaitlinkaiser@example.com    2017-08-03
4       Lori      Todd    Male  buchananmanuel@example.net    1938-12-01

             Job Title
0     Games developer
1      Phytotherapist
2           Homeopath
3   Market researcher
4  Veterinary surgeon
```

Dopo aver creato o caricato un DataFrame, esaminare e riassumere i dati è un passo importante per comprendere il dataset. Pandas fornisce varie funzioni che aiutano a visualizzare e analizzare i dati. Ad esempio:

- head(): visualizza le prime n righe del DataFrame (il valore predefinito è 5 righe)
- tail(): visualizza le ultime n righe del DataFrame (il valore predefinito è 5 righe)
- info(): questo metodo fornisce un riepilogo conciso del DataFrame, compreso il numero di voci non nulle, i nomi delle colonne e i tipi di dati

Vediamo un esempio che illustra l'uso dei metodi .head(), .tail() e .info():

[7]:
```
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
        'Age': [24, 27, 22, 32, 29],
        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix']}

df = pd.DataFrame(data)

# View the first 3 rows of the DataFrame
print("First 3 rows using head():")
```

```
print(df.head(3))

# View the last 2 rows of the DataFrame
print("\nLast 2 rows using tail():")
print(df.tail(2))

# Get a concise summary of the DataFrame
print("\nDataFrame summary using info():")
df.info()
```

```
First 3 rows using head():
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago

Last 2 rows using tail():
    Name  Age     City
3  David   32  Houston
4    Eva   29  Phoenix

DataFrame summary using info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    5 non-null      object
 1   Age     5 non-null      int64
 2   City    5 non-null      object
dtypes: int64(1), object(2)
memory usage: 252.0+ bytes
```

1.4 Selezione dei dati

La selezione dei dati in Pandas si riferisce al processo di accesso e selezione dei dati da un DataFrame o una serie Pandas. Esistono diversi modi per farlo. Per l'utilizzo di *loc()* ed *iloc()* si rimanda a tutorial specifici.

Esempio: indicizzazione di base (selezione di una singola colonna) con l'uso dell'operatore [ ]:

[8]:
```
data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}
df = pd.DataFrame(data)

# Select a single column
age_column = df['Age']
print(age_column)
```

```
0    25
```

```
1    30
2    35
Name: Age, dtype: int64
```

Un altro modo per filtrare i dati è quello di utilizzare una condizione booleana per selezionare soltanto i dati di nostro interesse. Ad esempio:

[9]:
```
data = {'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]}
df = pd.DataFrame(data)

# Filtering rows where Age is greater than 28
filtered_df = df[df['Age'] > 28]
print(filtered_df)
```

```
      Name  Age
1      Bob   30
2  Charlie   35
```

1.5 Gestione valori nulli

Capita spesso che all'interno di un dataset siano presenti dei valori nulli per alcuni record su specifici attributi. Ogni caso è a sé e bisogna decidere come gestire l'assenza di dati valutando caso per caso.

In questo esempio andremo a caricare un dataset che contiene valori nulli in svariati punti. Esclusivamente ai fini di questo tutorial, andremo a riempire quei vuoti utilizzando diverse misurazioni effettuate sulla colonna dove i dati sono assenti.

Andremo a sostituire i valori mancanti nella colonna *quantità* con la **media**, nella colonna *prezzo* con la **mediana**, nella colonna *Acquistato* con la **deviazione standard**. La colonna *Mattina* con il **valore minimo** in quella colonna. La colonna *Pomeriggio* con il **valore massimo** in quella colonna.

[12]:
```
# loading data set
data = pd.read_csv('./data/items.csv')

# display the data
print(data)

print('\nReplacing NaN Values...\n')

# replacing missing values in quantity
# column with mean of that column
data['quantity'] = data['quantity'].fillna(data['quantity'].mean())

# replacing missing values in price column
# with median of that column
data['price'] = data['price'].fillna(data['price'].median())

# replacing missing values in bought column with
```

```
# standard deviation of that column
data['bought'] = data['bought'].fillna(data['bought'].std())

# replacing missing values in forenoon  column with
# minimum number of that column
data['forenoon'] = data['forenoon'].fillna(data['forenoon'].min())

# replacing missing values in afternoon  column with
# maximum number of that column
data['afternoon'] = data['afternoon'].fillna(data['afternoon'].max())

print('\nNaN values replaced!\n')

print(data)
```

```
   id    item  quantity  price  bought  forenoon  afternoon
0   1    milk       2.0   67.0   672.0     456.0        NaN
1   2   sugar       1.0    NaN   453.0     234.0        NaN
2   3   chips       NaN   45.0   456.0     322.0        NaN
3   4  coffee       2.0   45.0   672.0     564.0        NaN
4   5    meat       4.0   56.0   786.0     221.0        NaN
5   6  chocos       3.0    NaN   345.0       NaN      213.0
6   7   juice       1.0   78.0   765.0       NaN      344.0
7   8     jam       NaN   65.0   665.0       NaN      333.0
8   9   bread       3.0    NaN     NaN       NaN      567.0
9  10  butter       4.0    NaN     NaN       NaN      322.0

Replacing NaN Values…


NaN values replaced!

   id    item  quantity  price      bought  forenoon  afternoon
0   1    milk       2.0   67.0  672.000000     456.0      567.0
1   2   sugar       1.0   60.5  453.000000     234.0      567.0
2   3   chips       2.5   45.0  456.000000     322.0      567.0
3   4  coffee       2.0   45.0  672.000000     564.0      567.0
4   5    meat       4.0   56.0  786.000000     221.0      567.0
5   6  chocos       3.0   60.5  345.000000     221.0      213.0
6   7   juice       1.0   78.0  765.000000     221.0      344.0
7   8     jam       2.5   65.0  665.000000     221.0      333.0
8   9   bread       3.0   60.5  162.022706     221.0      567.0
9  10  butter       4.0   60.5  162.022706     221.0      322.0
```

[ ]: