Web scraping: estrarre dati da siti per creare dataset

DI MASSIMO VAILATI

03/10/2025

PROGRAMMAZIONE WEB



Il **web scraping** è una tecnica che permette di **estrarre automaticamente dati da siti web** e **trasformarli in dataset utilizzabili per analisi e progetti**. Utilizzato in ricerca, business e media, consente di raccogliere informazioni aggiornate quando non esistono API ufficiali.

Tuttavia, richiede attenzione a privacy, termini d'uso e sostenibilità tecnica. In ambito didattico è uno strumento prezioso per imparare a esplorare l'HTML, comprendere il DOM e creare dataset personalizzati. I limiti principali riguardano la fragilità agli aggiornamenti dei siti e la necessità di pulire e strutturare i dati raccolti.

Viviamo in un'epoca in cui i dati rappresentano la linfa vitale dell'innovazione. Dalle ricerche accademiche alle start-up tecnologiche, la disponibilità di dataset ben strutturati è spesso la chiave per progetti di successo.

Il **web scraping** è il processo di estrazione automatica di dati da pagine web tramite software o script. Questi strumenti navigano sui siti, individuano le informazioni rilevanti (testo, immagini, tabelle) e le organizzano in formati facilmente analizzabili, come file CSV o database. Si tratta di una tecnica essenziale per chiunque abbia bisogno di grandi quantità di dati che non sono resi disponibili in modo strutturato direttamente dai siti web.

Molti settori fanno affidamento sul web scraping per ottenere dati aggiornati: ricerca scientifica, business intelligence, media e comunicazione. Senza il web scraping, l'accesso a molte di queste informazioni sarebbe lento, frammentario e spesso impossibile da gestire manualmente.

Estrarre dati dal web non è solo una questione tecnica: chi effettua scraping deve interrogarsi su questioni come il rispetto della privacy, la proprietà intellettuale, il carico sui server e la trasparenza delle intenzioni.

- Rispetto dei termini d'uso: molti siti web specificano nelle loro policy le condizioni per l'utilizzo dei dati e vietano espressamente lo scraping non autorizzato.
- Privacy degli utenti: i dati personali, come indirizzi email, numeri di telefono o informazioni sensibili, non dovrebbero mai essere raccolti senza consenso esplicito.
- Sostenibilità tecnica: un uso eccessivo di scraping può sovraccaricare i server dei siti target, creando disservizi e impatti economici.

Gli sviluppatori e i data scientist possono adottare alcune buone pratiche per garantire un approccio trasparente e responsabile:

- Leggere e rispettare il file robots.txt dei siti web, che indica le aree accessibili o vietate ai bot.
- Limitare la frequenza delle richieste per non sovraccaricare i server.
- Citare sempre la fonte dei dati raccolti e, dove possibile, chiedere il permesso per il loro utilizzo.
- Preferire l'uso di API ufficiali se disponibili, poiché offrono dati in modo strutturato e nel rispetto delle policy del servizio.

Uno dei principali punti di forza del web scraping è che può essere utilizzato anche quando non esistono API pubbliche disponibili. Per questo motivo, il web scraping è particolarmente utile in ambito didattico: consente agli studenti di lavorare con dati reali, non strutturati, e di creare dataset su misura per i propri progetti. È anche un ottimo esercizio per imparare a esplorare il codice HTML di una pagina, comprendere la struttura del DOM e applicare tecniche di parsing.

Tuttavia, ci sono diversi limiti da tenere presenti. Prima di tutto, la struttura di una pagina web può cambiare in qualsiasi momento: se il sito modifica anche solo leggermente l'HTML, lo scraper può smettere di funzionare. Questo lo rende uno strumento relativamente fragile rispetto alle API.

Inoltre, il processo di scraping può essere più lento e meno efficiente, soprattutto se si devono caricare molte pagine o navigare tra elementi complessi. Infine, l'analisi dei dati estratti richiede un lavoro di parsing spesso articolato, per pulire, strutturare e validare le informazioni raccolte. Questo può diventare complicato se la pagina contiene dati non ben formattati, o se si lavora con strutture HTML annidate.

Attività pratica n.1

Raccogliere dati delle previsioni meteo di una città

L'obiettivo è raccoglie le **previsioni meteo di una città per i prossimi 5 giorni** dal sito *www.meteo.it* e salvare i dati in un file CSV.

Innanzitutto, occorre **esplorare l'HTML** con gli strumenti per sviluppatore del browser che permettono di visualizzare il codice HTML organizzato in tempo reale. In questo modo è possibile vedere **quali tag** contengono le informazioni che vogliamo estrarre, capire come sono nidificati (es. `<div>`, `<span>`, `<a>`, `<img>`), e individuare classi, ID o attributi utili per selezionare i dati con precisione.

Apriamo la pagina https://www.meteo.it/meteo/milano, clicchiamo col tasto destro del mouse sull'icona della previsione generale per "domani" e apriamo l'analizzatore dal menu contestuale (su Chrome è la voce Esamina).

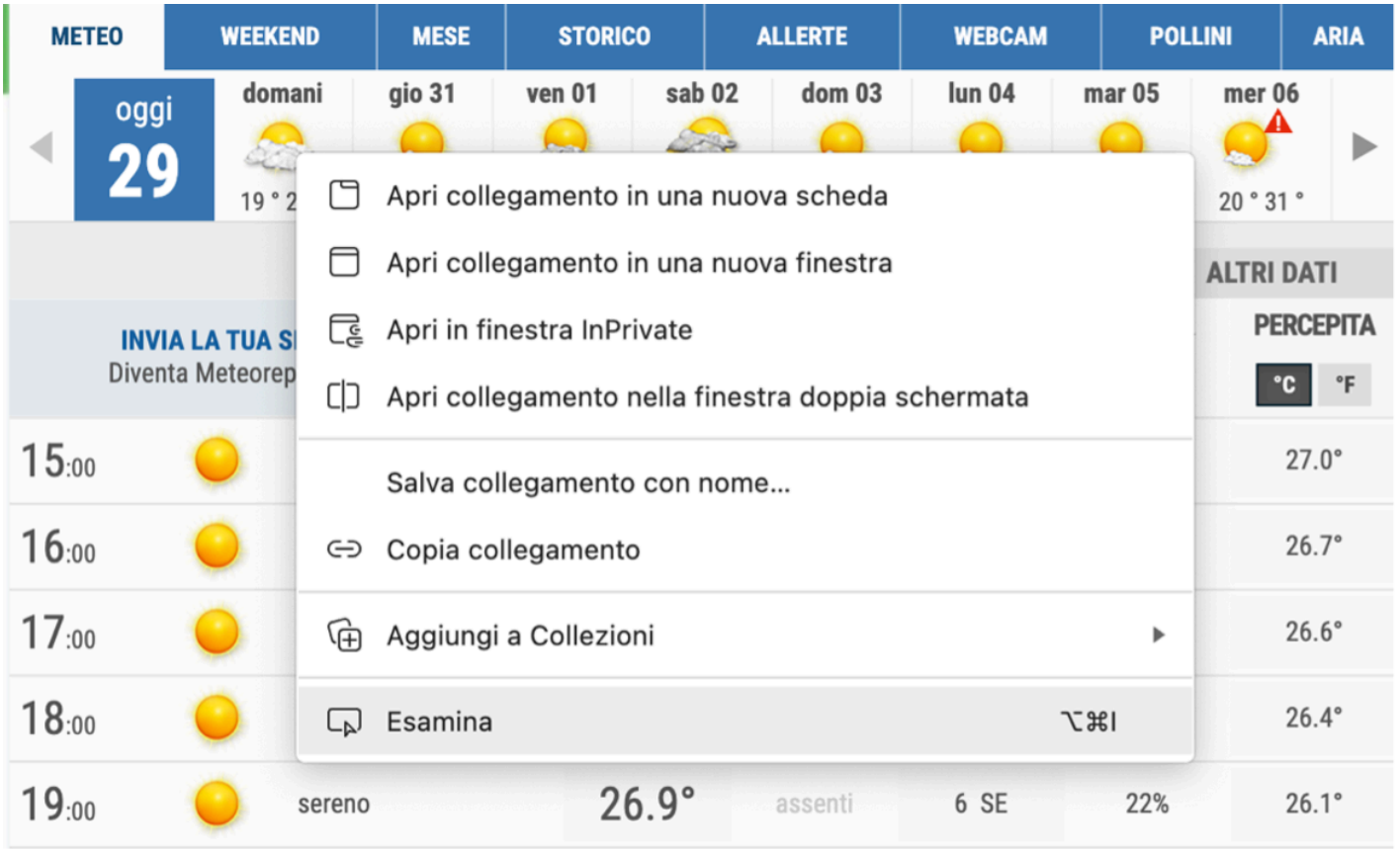


Fig.1

Il codice HTML della pagina presenta la seguente struttura:



Fig.2

Esaminando il codice HTML si osserva come il dato relativo ad un certo giorno inizia con il link `href="/meteo/milano/N"` (conNda 1 a 5) di un tag `<a>`, il giorno è descritto nel tag `<div>` seguente, e la previsione è sintetizzata nell'attributo alt del tag `<img>` successivo.

Per eseguire lo scraping scriviamo un programma in Python utilizzando la libreria **BeautifulSoup** progettata per facilitare l'estrazione di dati da documenti HTML e XML.

Ecco un esempio pratico d'uso della libreria in cui si scarica la home page del sito **www.example.com**, si fa il parse dell'html, infine si cercano tutti tag h2 e si stampa per ciascuno il relativo testo a schermo.

```
import requests
from bs4 import BeautifulSoup

url = "https://www.example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

```
titoli = soup.find_all('h2')
for titolo in titoli:
    print(titolo.text)
```

La libreria si può installare con il comando:

```
pip install requests beautifulsoup4
```

Ecco ora il codice Python, sufficientemente commentato, del programma di scraping per il nostro caso:

```
import requests
from bs4 import BeautifulSoup
import csv
import sys
import unicodedata

def normalizza_nome_citta(nome):
    # Rimuove accenti e caratteri speciali
    nome = unicodedata.normalize('NFKD', nome)
    nome = nome.encode('ASCII', 'ignore').decode('utf-8')
    return nome.lower().replace(" ", "-")

def tronca_giorno(testo):
    testo = testo.strip()
    if testo.lower().startswith("domani"):
        return testo[:6]
    else:
        return testo[:5]

def main():
    # Il nome della città viene letto da riga di comando
    # per generalizzare il codice
    if len(sys.argv) < 2:
```

```
        print("Uso: python meteo_scraper.py <nome_città>")
        sys.exit(1)

    citta_input = sys.argv[1]
    citta_url = normalizza_nome_citta(citta_input)

    # Prepara l'URL da interrogare e imposta un User-Agent per simulare un
browser
    # (alcuni siti bloccano bot automatici).
    url = f"https://www.3bmeteo.com/meteo/{citta_url}"
    headers = {'User-Agent': 'Mozilla/5.0'}

    print(f"Scarico previsioni per: {citta_input} ({url})")

    #Esegue la richiesta HTTP e verifica che la risposta sia valida (status 200).
    # In caso di errore, stampa un messaggio e termina.
    response = requests.get(url, headers=headers)
    if response.status_code != 200:
        print(f"Errore: impossibile accedere alla pagina {url}")
        sys.exit(1)

    # Analizza il contenuto HTML della pagina con BeautifulSoup,
    # e inizializza una lista per raccogliere le previsioni.
    soup = BeautifulSoup(response.text, 'html.parser')
    previsioni = []

    for n in range(1, 6):
        # Cerca i tag <a> con href simile a /meteo/nome-citta/1, /2, ... /5
        # che rappresentano i link dei prossimi 5 giorni.
        giorno_tag = soup.find("a", href=f"/meteo/{citta_url}/{n}")
        if giorno_tag:
   # Estrae il testo (nome giorno), lo tronca con tronca_giorno,
            # cerca l'immagine successiva (<img>) da cui prende l'attributo alt,
            # che contiene la descrizione meteo (es. "poco nuvoloso"),
            # Aggiunge tutto alla lista previsioni.
            giorno_text = giorno_tag.get_text(strip=True)
            giorno_troncato = tronca_giorno(giorno_text)
            img_tag = giorno_tag.find_next("img")
            descrizione = img_tag["alt"] if img_tag and img_tag.has_attr("alt") else
"Previsione non trovata"
```

```
            previsioni.append([giorno_troncato, descrizione])
        else:
            previsioni.append([f"G{n}", "Non trovato"])

    # Scrivi su file CSV
    nome_file = f"previsioni_{citta_url}.csv"
    with open(nome_file, "w", newline='', encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["Giorno", "Previsione"])
        writer.writerows(previsioni)

    print(f"Previsioni salvate in: {nome_file}")

if __name__ == "__main__":
    main()
```

Per eseguire il programma dobbiamo specificare la città che ci interessa (ad esempio milano):

```
python meteo_scraper.py milano
```

Ecco l'output del programma:

```
Scarico previsioni per: milano (https://www.3bmeteo.com/meteo/milano)
Previsioni salvate in: previsioni_milano.csv
```

e il contenuto del file *previsioni_milano.csv* creato:

```
Giorno,Previsione
domani,nubi sparse
gio31,poco nuvoloso
ven01,possibili temporali
```

```
sab02,nubi sparse e temporali
dom03,poco nuvoloso
```