Webscraping für Datenjournalist:innen

Hands-on workshop zum Extrahieren von Daten aus dem Internet

Kai-Robin Lange[1], Lisa-Marie Eckardt[12], Henrike Weinert[1], Katja Ickstadt[1]

[1]Technische Universität Dortmund
[2]Westdeutscher Rundfunk

05.09.2025

Data Competence Network @TU Dortmund

dortmund university

Digital Humanities Ruhr TUDO

Wichtige Downloads

Alles Wichtige zu diesem Workshop findet ihr unter
`github.com/K-RLange/SciCAR_Webscraping`

Python

- `requests` & `BeautifulSoup` – Anfragen und parsen von HTML
- `selenium` & `pyautogui` – Browser fernsteuern

R

- `httr` & `rvest` – HTTP-Handling & einfache Syntax zum Auslesen von HTML
- `RSelenium` – ähnlich wie Python Selenium

Zeitplan

Workshop-Plan

1. Einführung: Motivation, Definition, rechtliche Aspekte, Grundlagen (20 Min.)
2. Praktischer Teil BeautifulSoup und/oder Selenium (40 Min.)

Zielgruppe

- Anfänger:innen mit Grundkenntnissen in Python oder R
- Fortgeschrittene mit Interesse an technischen Hürden

Was ist Webscraping?

Definition

Webscraping bedeutet: *automatisiertes Auslesen von Inhalten aus Webseiten*. Sowohl strukturierte (JSON, CSV,...), als auf unstrukturierte Formen (txt) sind möglich.

Warum ist das nützlich für Datenjournalismus?

- Viele Daten sind nicht als offene Datensätze verfügbar
- Webseiten bieten Informationen, die manuell kaum sammelbar sind
- Automatisierte Datenextraktion spart Zeit und Ressourcen
- Beispiele:
  - Live-Daten (bspw. COVID-19 Fälle)
  - Gerichtsurteile extrahieren
  - Preis- und Angebotsentwicklung dokumentieren

Rechtliche Aspekte (Überblick)

Grundprinzipien

- **AGBs:** prüfen, ob Scraping untersagt ist
- **Urheberrecht:** Texte und Datenbanken können geschützt sein
- **Datenschutz:** personenbezogene Daten besonders sensibel
- **Robots.txt:** beschreibt gewünschtes Verhalten – nicht rechtsverbindlich

Best Practices

- Nur notwendige Daten sammeln
- Quellen transparent dokumentieren
- Bei Zweifeln: rechtliche Beratung einholen
- Abfragen takten (*Rate Limiting*) – keine Server überlasten

Typische Herausforderungen

Technische Probleme

- Unterschiedliche HTML-Strukturen je nach Seite
- Dynamisch nachgeladene Inhalte via JavaScript
- Anti-Scraping-Techniken: Captchas, IP-Blocking

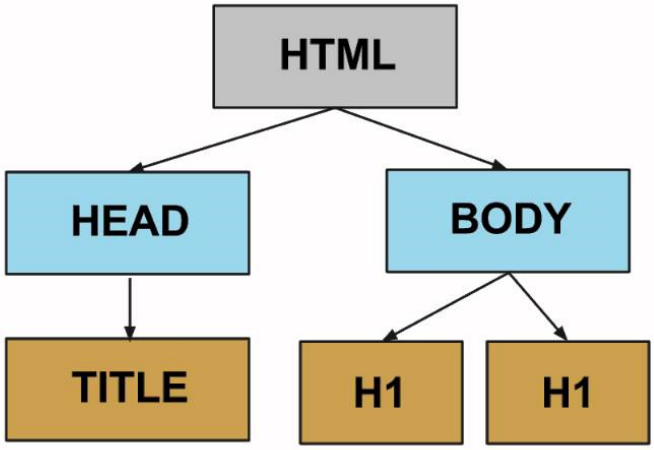
Organisatorische und rechtliche Herausforderungen

- Seitenstrukturen können sich ändern: Fehler im Scraping erkennen und aktualisieren
- Viele Scraper (bspw. `Selenium`) werden häufig geupdated
- Datenqualität prüfen: nicht alle Inhalte sind sauber strukturiert
- Daten speichern und versionieren (z. B. Git, SQL-Datenbanken)

Übersicht: HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>

    <body>
        <h1></h1>
        <h1></h1>
    </body>
</html>
```



Tools für Webscraping

Python

- `requests` & `BeautifulSoup` – Anfragen und parsen von HTML
- `selenium` & `pyautogui` – Browser fernsteuern

R

- `httr` & `rvest` – HTTP-Handling & einfache Syntax zum Auslesen von HTML
- `RSelenium` – ähnlich wie Python Selenium

Weitere Tools

- Browser-DevTools (Element-Inspektor, Netzwerk-Tab)
- regex101 für reguläre Ausdrücke
- APIs: manchmal ist der offizielle Weg besser als Scraping

Praktischer Teil 1: BeautifulSoup

Beispiel

Wikipedia: Netzwerk Recherche
Ziel: Strukturierte Infos aus der Infobox extrahieren

Schritte

1. HTTP-Request: Seite herunterladen
2. HTML parsen: `BeautifulSoup`
3. Struktur untersuchen: mit Browser-Inspektor
4. Elemente finden: `find`, `find_all`
5. Ergebnisse weiterverarbeiten: CSV/Excel speichern

Praktischer Teil 2: Interaktive Seiten

Problem

Webseiten nutzen oft JavaScript, um Inhalte erst bei Bedarf zu laden:

- Klick auf „Mehr anzeigen“ oder scrollen
- Dropdown-Menüs

Lösung: Browser-Automatisierung

- `Selenium`: Standardtool, steuert echte Browser
- `Playwright`: Moderner, schneller, paralleles Scraping
- Nachteil: Höhere Systemlast, langsamer als requests

Praktischer Teil 2: Interaktive Seiten

Problem

Webseiten nutzen oft JavaScript, um Inhalte erst bei Bedarf zu laden:

- Klick auf „Mehr anzeigen“ oder scrollen
- Dropdown-Menüs

Lösung: Browser-Automatisierung

- `Selenium`: Standardtool, steuert echte Browser
- `Playwright`: Moderner, schneller, paralleles Scraping
- Nachteil: Höhere Systemlast, langsamer als requests

Beispiel

Quotes to Scrape, Ziel: Alle Zitate scrapen und ausgeben.