# Intro To Web Scraping in Python

## Intro

### Data Never Sleeps

- The total volume of data created and stored reached 64.2 ZB in 2020 and is expected to grow to more than 180 zettabytes by 2025 (Statista, 2021)
- Every minute, users worldwide conduct 5.9 million searches on Google, share 437,000 Tweets, and upload 500 hours of videos on YouTube. (DOMO, 2022)
- A considerable part of this data is **publicly available online** and offers valuable insights for economic and business research (e.g., quantify consumption, measure prices, track consumer behaviors, etc.)
- Example: Growing use of web data across the top 5 marketing journals (Boegershausen et al., 2022)

### Collecting Online Data

- **Problem**: Gathering data dispersed throughout the Internet manually is usually unfeasible or incredibly time-consuming.
- **Solution**: Researchers can automate the data collection process with the use of:
  - **Application Programming Interfaces (APIs)**: often unavailable or expensive
  - **Web Scraping**

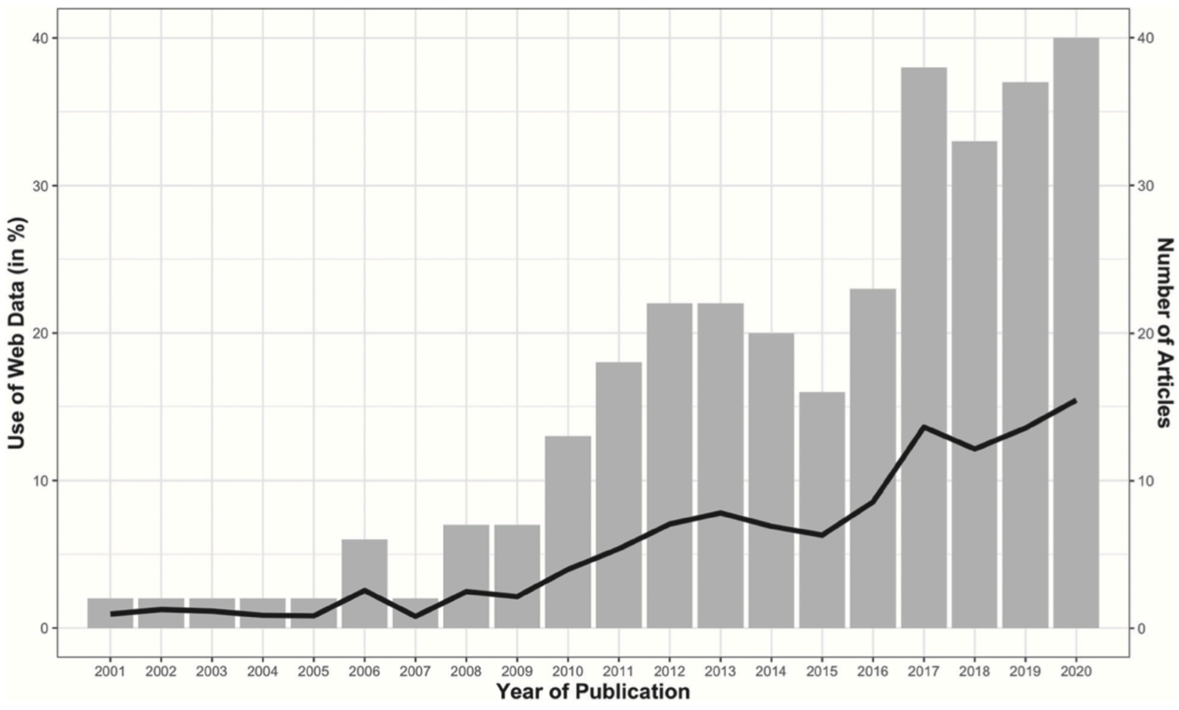


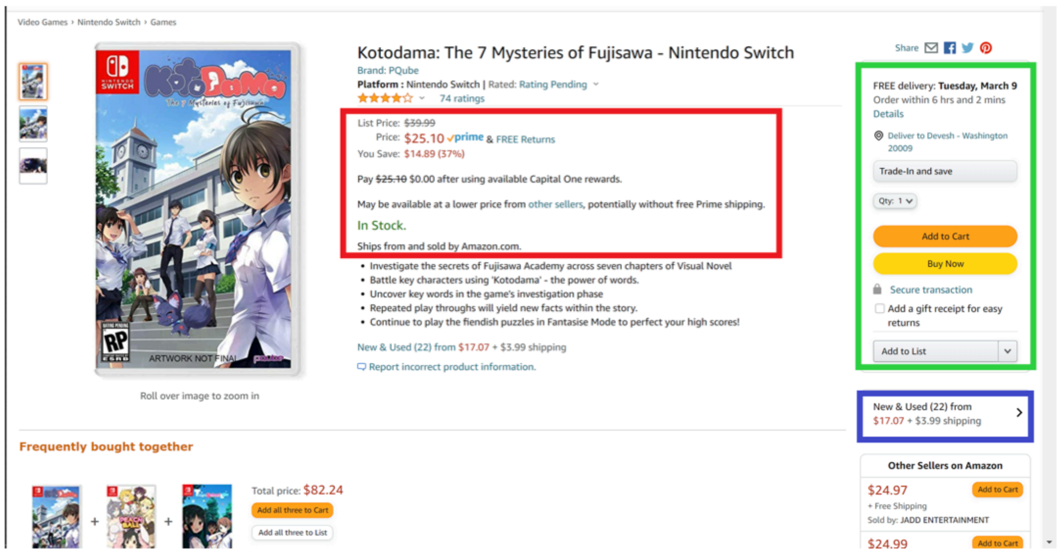
Figure 1: convert notebook to web app

## What is Web Scraping?

- **Web scraping** is the automated process of extracting information from websites using specialized programs or scripts.
- Key advantages:
  - **Enhanced data accessibility**: no gatekeepers, cheaper and faster than traditional data collection methods
  - **Flexibility in data collection**: more control over various parameters, like frequency and granularity.
  - **Simplicity of Implementation**: powerful libraries available in many programming languages (e.g. Python, R, Julia)

## Some Examples

Raval (2023): Scraped information for hundreds of thousands of products on Amazon and used it to investigate what drives the BuyBox assignment.



Video Games › Nintendo Switch › Games

Kotodama: The 7 Mysteries of Fujisawa - Nintendo Switch
Brand: PQube
**Platform :** Nintendo Switch | Rated: Rating Pending
74 ratings

List Price: ~~$39.99~~
Price: $25.10 ✓prime & FREE Returns
You Save: $14.89 (37%)

Pay ~~$25.10~~ $0.00 after using available Capital One rewards.

May be available at a lower price from other sellers, potentially without free Prime shipping.

In Stock.

Ships from and sold by Amazon.com.

- Investigate the secrets of Fujisawa Academy across seven chapters of Visual Novel
- Battle key characters using 'Kotodama' - the power of words.
- Uncover key words in the game's investigation phase
- Repeated play throughs will yield new facts within the story.
- Continue to play the fiendish puzzles in Fantasise Mode to perfect your high scores!

New & Used (22) from $17.07 + $3.99 shipping
Report incorrect product information.

Roll over image to zoom in

Frequently bought together

Total price: $82.24

FREE delivery: **Tuesday, March 9**
Order within 6 hrs and 2 mins
Details

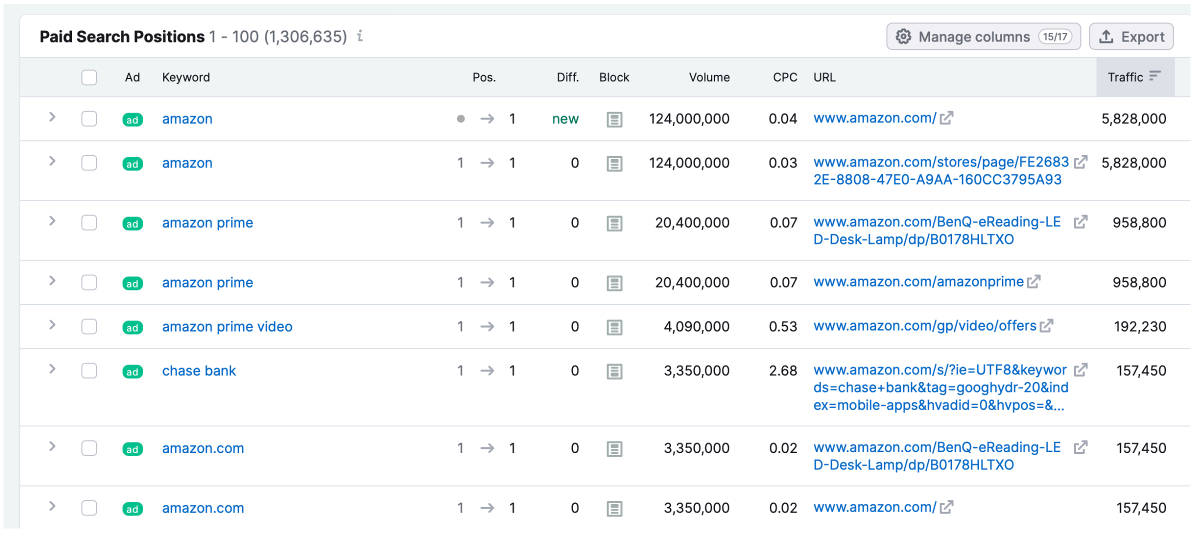
Deliver to Devesh - Washington 20009

New & Used (22) from
$17.07 + $3.99 shipping

Other Sellers on Amazon
$24.97 + Free Shipping
Sold by: JADD ENTERTAINMENT
$24.99

(a) Product Detail Page

Decarolis, Rovigatti (2021) scraped data on nearly 40 million Google keyword auctions from Semrush.com and applied ML techniques to define advertising markets.



**Paid Search Positions** 1 - 100 (1,306,635)

Manage columns 15/17 | Export

| Ad | Keyword | Pos. | Diff. | Block | Volume | CPC | URL | Traffic |
|---|---|---|---|---|---|---|---|---|
| ad | amazon | ● → 1 | new | ▤ | 124,000,000 | 0.04 | www.amazon.com/ | 5,828,000 |
| ad | amazon | 1 → 1 | 0 | ▤ | 124,000,000 | 0.03 | www.amazon.com/stores/page/FE26832E-8808-47E0-A9AA-160CC3795A93 | 5,828,000 |
| ad | amazon prime | 1 → 1 | 0 | ▤ | 20,400,000 | 0.07 | www.amazon.com/BenQ-eReading-LED-Desk-Lamp/dp/B0178HLTXO | 958,800 |
| ad | amazon prime | 1 → 1 | 0 | ▤ | 20,400,000 | 0.07 | www.amazon.com/amazonprime | 958,800 |
| ad | amazon prime video | 1 → 1 | 0 | ▤ | 4,090,000 | 0.53 | www.amazon.com/gp/video/offers | 192,230 |
| ad | chase bank | 1 → 1 | 0 | ▤ | 3,350,000 | 2.68 | www.amazon.com/s/?ie=UTF8&keywords=chase+bank&tag=googhydr-20&index=mobile-apps&hvadid=0&hvpos=&... | 157,450 |
| ad | amazon.com | 1 → 1 | 0 | ▤ | 3,350,000 | 0.02 | www.amazon.com/BenQ-eReading-LED-Desk-Lamp/dp/B0178HLTXO | 157,450 |
| ad | amazon.com | 1 → 1 | 0 | ▤ | 3,350,000 | 0.02 | www.amazon.com/ | 157,450 |

### Our goals for today

1. Familiarize with the basics of web scraping
2. Practice with main libraries for web scraping in Python
3. Discuss how to approach web scraping in a responsible and smart way

## The requests package

### HTTP requests

- **HTTP (Hypertext Transfer Protocol)** is the protocol used to transfer data over the web.
- It operates on a request-response model, where:
    1. a client (e.g., a web browser) sends a request to a server
    2. the server responds with the requested information.
- It defines various methods that indicate the action the client wants to perform on a resource. Most useful for web scraping:
    - the **GET** method retrieves data
    - the **POST** method submits data
- `requests` is a library that allows to handle HTTP requests in Python

### Examples

```
# EXAMPLE OF GET REQUEST
import requests

# GET request example
response = requests.get(
    "https://jsonplaceholder.typicode.com/albums"
    )

# Check the status code (200 is good, 400-ish is bad)
print("Status code:", response.status_code)

# the content of the response can be formatted as
# a list of dictionaries using the .json() method
print(response.json()[0])
```

```
Status code: 200
{'userId': 1, 'id': 1, 'title': 'quidem molestiae enim'}
```

```
# EXAMPLE OF POST REQUEST
import json

# Define the headers for the POST request
headers = {
    'Content-type': 'application/json; charset=UTF-8',
}

# Define the data we want to attach (i.e. the payload)
data = { 'userId': 1, 'title': 'my title'}

# Send the POST request
response = requests.post(
    'https://jsonplaceholder.typicode.com/posts',
data=json.dumps(data), headers=headers)

# Print the newly added data
print(response.json())
```

```
{'userId': 1, 'title': 'my title', 'id': 101}
```

### Main Use-Cases

We cover three use-cases of the requests package:

- HTML-based Web Scraping
- XHR-based Web Scraping
- Automated files download

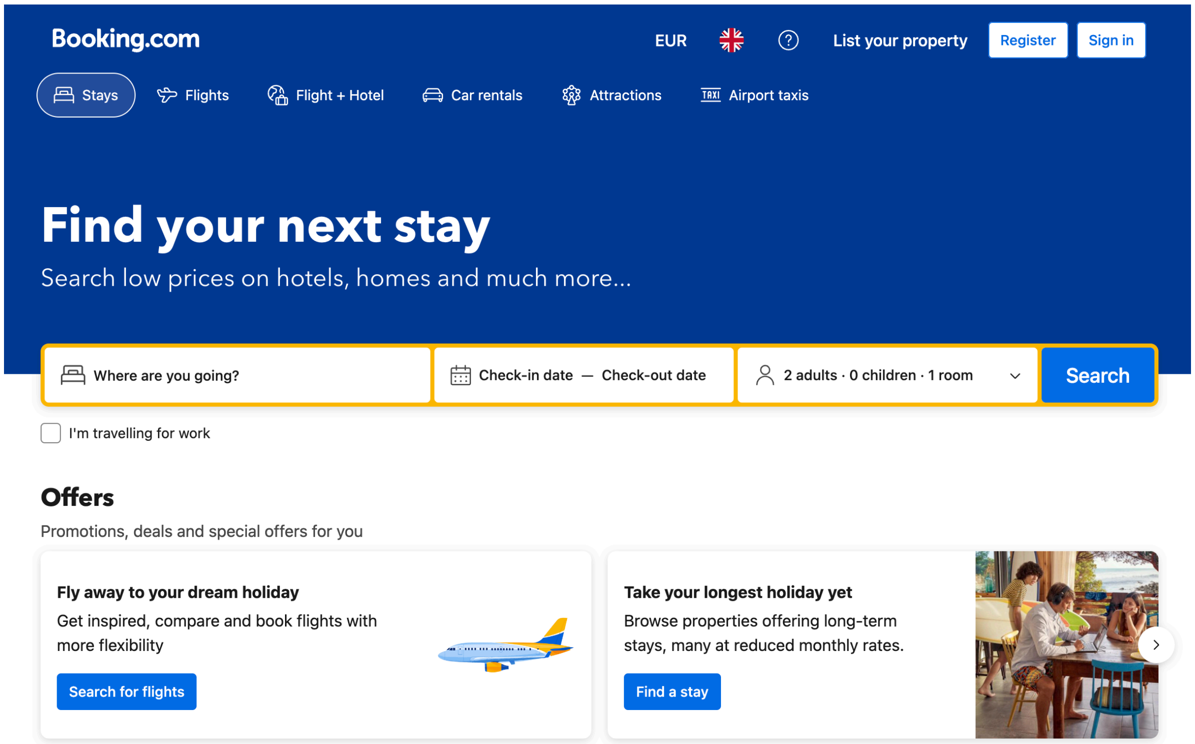
But there is much more:

- Authentication and logins
- Use of official APIs
- IP Rotation

## 1. HTML-based Web Scraping

### HTML: The Language of the Web

When you order something online...



Find your next stay

Search low prices on hotels, homes and much more...

Where are you going?
Check-in date — Check-out date
2 adults · 0 children · 1 room
Search
I'm travelling for work

Offers

Promotions, deals and special offers for you

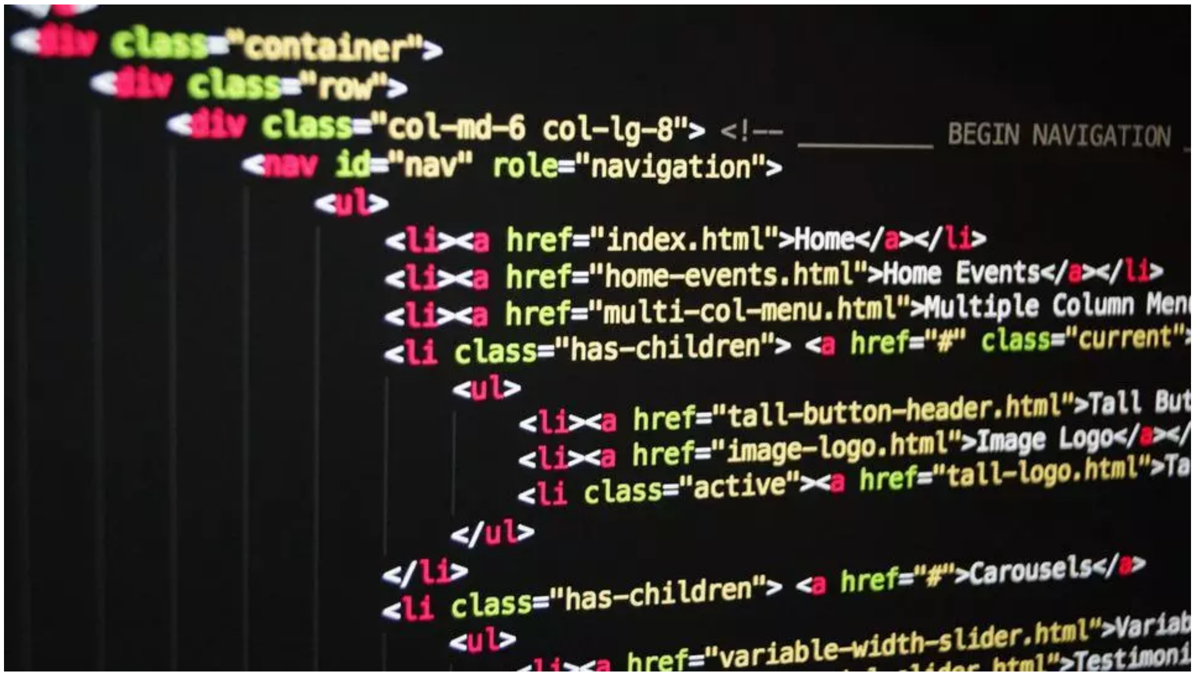
Fly away to your dream holiday

Get inspired, compare and book flights with more flexibility

Take your longest holiday yet

Browse properties offering long-term stays, many at reduced monthly rates.

... when it arrives



```
<div class="container">
  <div class="row">
    <div class="col-md-6 col-lg-8"> <!-- __________ BEGIN NAVIGATION
      <nav id="nav" role="navigation">
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="home-events.html">Home Events</a></li>
          <li><a href="multi-col-menu.html">Multiple Column Men
          <li class="has-children"> <a href="#" class="current">
            <ul>
              <li><a href="tall-button-header.html">Tall But
              <li><a href="image-logo.html">Image Logo</a></
              <li class="active"><a href="tall-logo.html">Ta
            </ul>
          </li>
          <li class="has-children"> <a href="#">Carousels</a>
            <ul>
              <li><a href="variable-width-slider.html">Variab
```

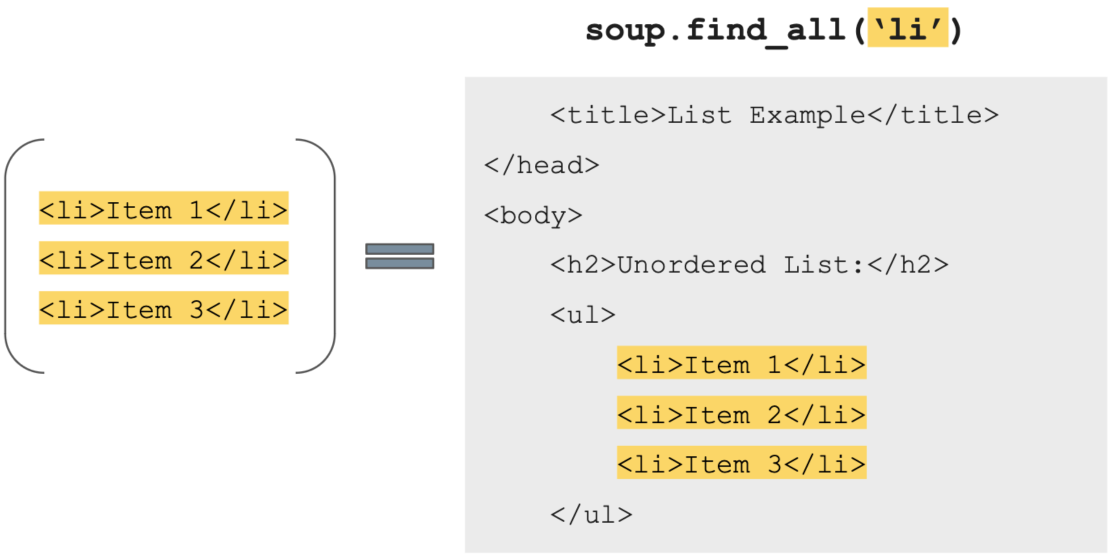
7

- **HTML (HyperText Markup Language)** is the standard language used for creating and structuring content of webpages.
- HTML documents are composed of **HTML elements**, which are defined by a start tag, some content, and an end tag.

  `<h1>My First Heading</h1>`
- **HTML tags** are like keywords which define how web browsers will format and display the content.
  - Some common tags are `li` for lists, `a` for links, and `table` for tables. See more tags

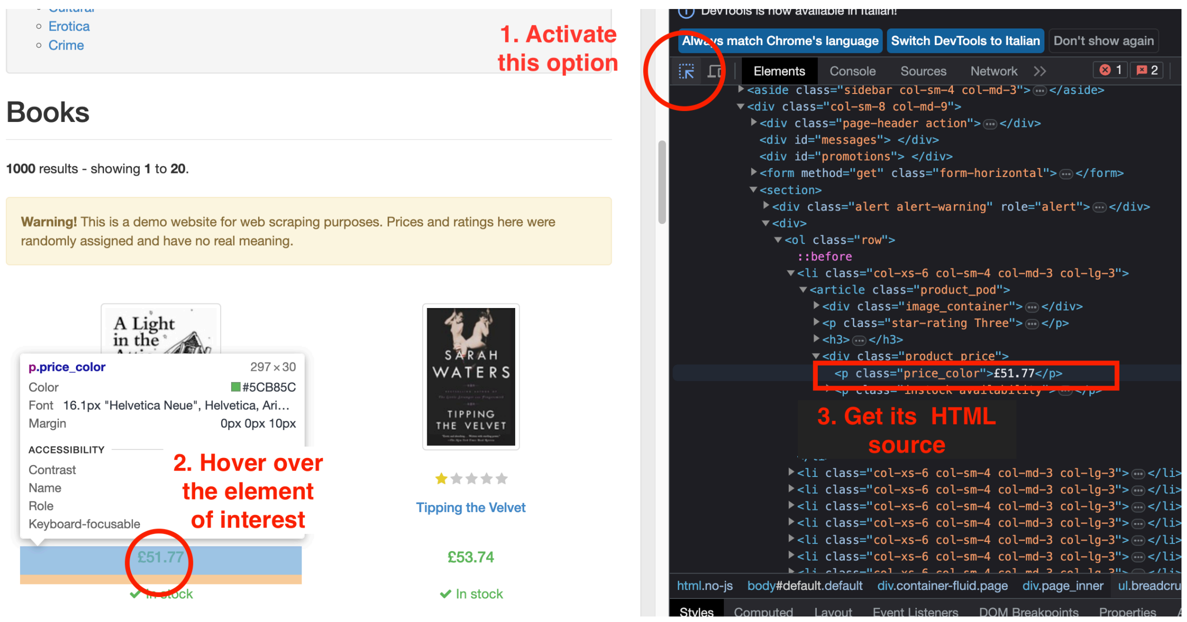
## HTML is a beautiful soup!

`beautifulsoup` is a Python package that can parse HTML content and extract information from its elements.



```
    <title>List Example</title>
</head>
<body>
    <h2>Unordered List:</h2>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
```

## How to Use Developer Tools


1. Activate this option
2. Hover over the element of interest
3. Get its HTML source
Erotica
Crime
Books
p.price_color
297 × 30
Color
🟩 #5CB85C
Font 16.1px "Helvetica Neue", Helvetica, Ari…
Margin 0px 0px 10px
ACCESSIBILITY
Contrast
Name
Role
Keyboard-focusable
£51.77
Tipping the Velvet
£53.74
✓ In stock
DevTools is now available in Italian!
Always match Chrome's language
Switch DevTools to Italian
Don't show again
Elements
Console
Sources
Network
html.no-js body#default.default div.container-fluid.page div.page_inner ul.breadcru
Styles Computed Layout Event Listeners DOM Breakpoints Properties

1000 results - showing 1 to 20.

Warning! This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.

```
<aside class="sidebar col-sm-4 col-md-3">…</aside>
<div class="col-sm-8 col-md-9">
  <div class="page-header action">…</div>
  <div id="messages"> </div>
  <div id="promotions"> </div>
  <form method="get" class="form-horizontal">…</form>
  <section>
    <div class="alert alert-warning" role="alert">…</div>
    <div>
      <ol class="row">
        ::before
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">
          <article class="product_pod">
            <div class="image_container">…</div>
            <p class="star-rating Three">…</p>
            <h3>…</h3>
            <div class="product_price">
              <p class="price_color">£51.77</p>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
        <li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">…</li>
```

## Bookstore Example

```python
from bs4 import BeautifulSoup
import pandas as pd
import os

# Define the target URL
target_url = "https://books.toscrape.com/"

# Request the page
page = requests.get(target_url)
```

```python
# Inspect the content of the page: it's the HTML source of the webpage
print(page.content[:30])
```

```
b'<!DOCTYPE html>\n<!--[if lt IE '
```

```python
# Parse the page
soup = BeautifulSoup(page.content, "html.parser")
```

```
# print(soup.prettify())
```

```
# Explore the HTML from the dev tools in Chrome
# and find the relevant elements
products = soup.find_all("article", class_="product_pod")
print('Number of products:', len(products))
```

```
Number of products: 20
```

```
# Extract product title and price
titles = []
prices = []

for p in products:
    titles.append(p.find("h3").find("a")["title"])
    prices.append(
            p.find("p", class_="price_color").text[1:])


# Build the dataframe
df = pd.DataFrame(
    {"title": titles, "price": prices}
)
df['price'] = df['price'].astype(float)

print(df.head())
```

```
                                 title  price
0                   A Light in the Attic  51.77
1                     Tipping the Velvet  53.74
2                             Soumission  50.10
3                          Sharp Objects  47.82
4  Sapiens: A Brief History of Humankind  54.23
```

```
# Create a folder and save the dataframe as a CSV file
if not os.path.exists("data"):
    os.makedirs("data")
```

```
# Store the dataframe as a CSV file
df.to_csv("data/books_example.csv", index=False)
```

Let's now scale up our scraping example to multiple book categories.

```
from tqdm import tqdm
from time import sleep
import random

# Get the full list of book categories
categories = soup.find("div", class_="side_categories")

# Get the full list of book categories
ul_element = soup.find("ul",
                       class_ = 'nav nav-list').find("ul")
li_elements = ul_element.find_all("li")
categories = [category.text.strip()
                for category in li_elements]

# # Print the first four categories
print(categories[:3])
```

```
['Travel', 'Mystery', 'Historical Fiction']
```

```
# Initialize a list to store the category dataframes
df_list = []

# Define the base url
base_url = "https://books.toscrape.com/catalogue/category/books/"

# Scrape the first page for the first 3 categories
for category, idx in tqdm(zip(categories[:3], range(2,5))):

    # Wait for 1 to 3 seconds before each request
    # (to avoid overloading the server)
    sleep(random.randint(1, 3))

    # Build the category url
    category_url = (
        base_url
        + category.lower().replace(' ', '-')
        + '_'
        + str(idx)
        + '/index.html'
    )

    # Request the page
    category_page = requests.get(category_url)

    # Check the status code
    if category_page.status_code != 200:
        print("Error with category:", category)
        continue

    # Parse the page
    category_soup = BeautifulSoup(category_page.content, "html.parser")

    # Find the relevant tag
    category_products = category_soup.find_all("article", class_="product_pod")

    # Extract product title
    category_titles = [p.find("h3").find("a")["title"] for p in category_products]

    # Extract product price
    category_prices = [
        p.find("p", class_="price_color").text[1:] for p in category_products
    ]

    # Extract product rating
    category_ratings = [
        p.find("p", class_="star-rating")["class"][1] for p in category_products
    ]

    # Build the dataframe
    category_df = pd.DataFrame(
        {"title": category_titles, "price": category_prices, "rating": category_ratings}
    )
    category_df["category"] = category
    category_df['price'] = category_df['price'].astype(float)
    df_list.append(category_df)
```

```
# Concatenate all the dataframes
categories_df = pd.concat(df_list)
print(categories_df.info())
print(categories_df.head())
categories_df.to_csv("data/books_example_categories.csv", index=False)
```

```
0it [00:00, ?it/s]3it [00:05,  1.82s/it]
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 51 entries, 0 to 19
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   title     51 non-null     object 
 1   price     51 non-null     float64
 2   rating    51 non-null     object 
 3   category  51 non-null     object 
dtypes: float64(1), object(3)
memory usage: 2.0+ KB
None
                                               title  price rating category
0                            It's Only the Himalayas  45.17    Two   Travel
1  Full Moon over Noah’s Ark: An Odyssey to Mount...  49.43   Four   Travel
2  See America: A Celebration of Our National Par...  48.87  Three   Travel
3  Vagabonding: An Uncommon Guide to the Art of L...  36.94    Two   Travel
4                               Under the Tuscan Sun  37.33  Three   Travel
```

```
# EXCERCISE
# Scrape the first 10 pages of books from the first 10 categories
# Hint: You can get the number of pages from the bottom of the webpage
# category_pages = category_soup.find('li', class_='current').text
```

### HTML-based Web Scraping: S&W

**Strengths**:

- Easy to implement and debug
- Faster than other approaches

- Scalable

**Weaknesses**:

- Could be unstable over time (source code changes)
- Cannot handle dynamically loaded content
- Cannot interact with the webpage (click, type, scroll, …)

## 2. XHR-based Web Scraping

### Intuition

Webpages can be thought of as a combination of structure and content.

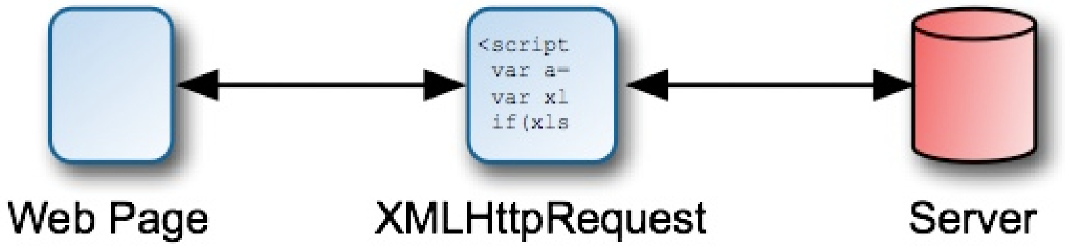
```html
<head>
    <title>List Example</title>
</head>
<body>
    <h2>Unordered List:</h2>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>
```

XHR-based web scraping allows to request the content directly, without caring about the structure.

```
<head>
    <title>             </title>             List Example
</head>
<body>
    <h2>             </h2>             Unordered List:
    <ul>
        <li>    </li>                         Item 1
        <li>    </li>                         Item 2
        <li>    </li>                         Item 3
    </ul>
```

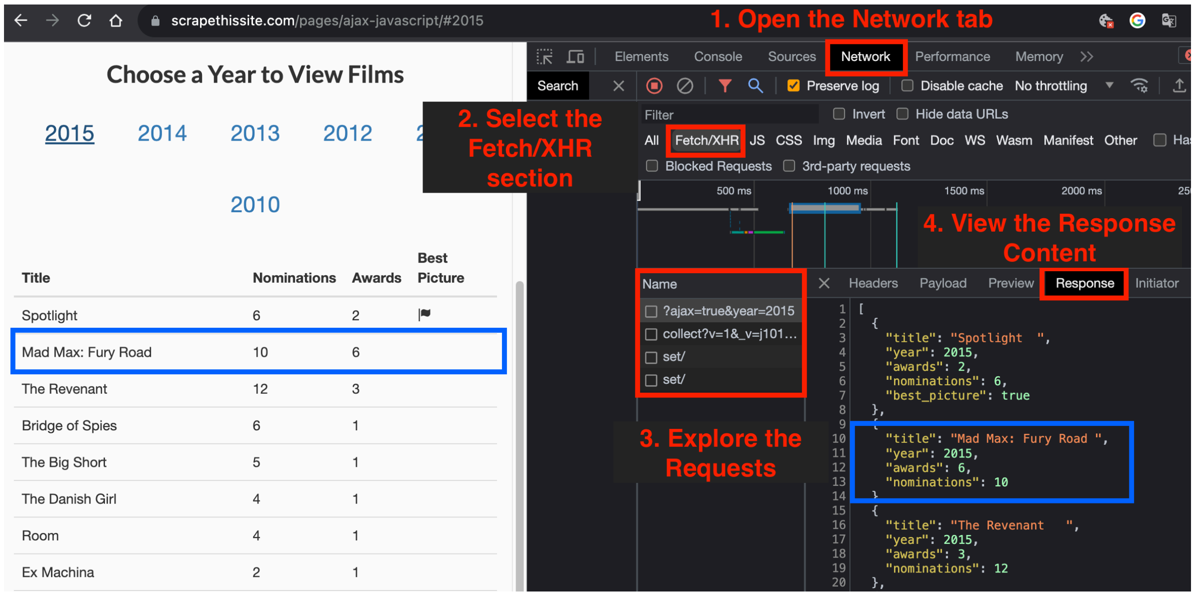
**Entering from the Backdoor**

This technique aims to intercept the XHR requests that the web page sends in the background to update its content without reloading the structure of the page.



OPEN MOVIES EXAMPLE

## Monitoring HXR Requests in Chrome



scrapethissite.com/pages/ajax-javascript/#2015

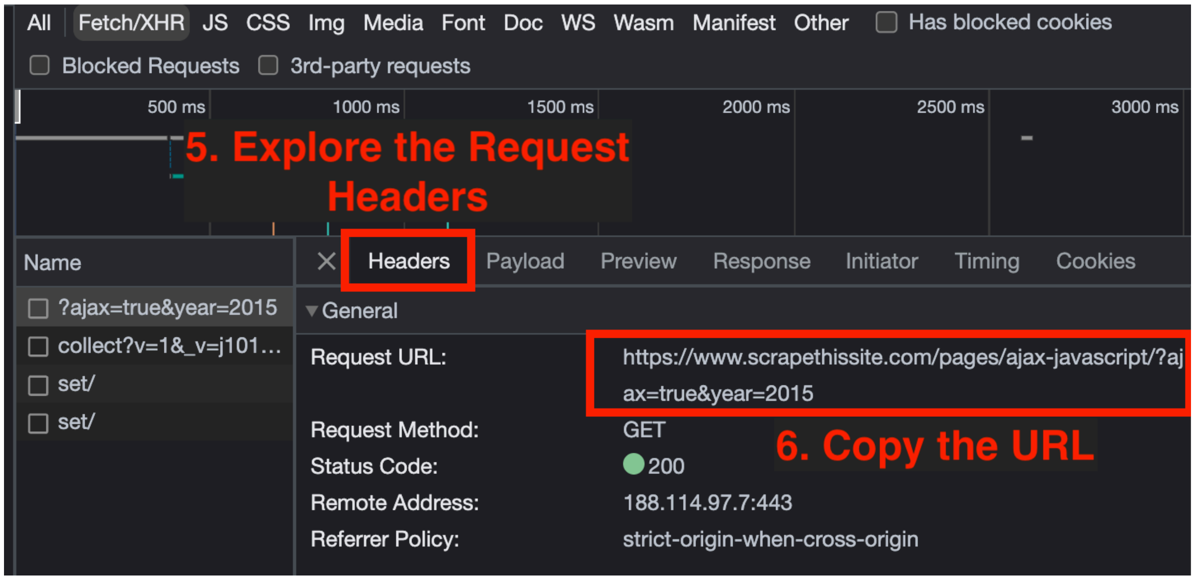
Choose a Year to View Films

2015 2014 2013 2012 2010

| Title | Nominations | Awards | Best Picture |
|---|---|---|---|
| Spotlight | 6 | 2 | ⚑ |
| Mad Max: Fury Road | 10 | 6 | |
| The Revenant | 12 | 3 | |
| Bridge of Spies | 6 | 1 | |
| The Big Short | 5 | 1 | |
| The Danish Girl | 4 | 1 | |
| Room | 4 | 1 | |
| Ex Machina | 2 | 1 | |

Name
- ?ajax=true&year=2015
- collect?v=1&_v=j101…
- set/
- set/

```
[
  {
    "title": "Spotlight  ",
    "year": 2015,
    "awards": 2,
    "nominations": 6,
    "best_picture": true
  },
  {
    "title": "Mad Max: Fury Road ",
    "year": 2015,
    "awards": 6,
    "nominations": 10
  },
  {
    "title": "The Revenant   ",
    "year": 2015,
    "awards": 3,
    "nominations": 12
  },
```



Name
- ?ajax=true&year=2015
- collect?v=1&_v=j101…
- set/
- set/

General

| | |
|---|---|
| Request URL: | https://www.scrapethissite.com/pages/ajax-javascript/?ajax=true&year=2015 |
| Request Method: | GET |
| Status Code: | ● 200 |
| Remote Address: | 188.114.97.7:443 |
| Referrer Policy: | strict-origin-when-cross-origin |

### Oscar Winning Films Example

```python
# Define the URL
base_url = 'https://www.scrapethissite.com/pages/ajax-javascript/'
parameters = '?ajax=true&year=2015'
url = base_url + parameters

# Read the headers from the header/headers.json file
with open('files/headers/header.json', 'r') as f:
    headers = json.load(f)

# Send the GET request
response = requests.get(url, headers=headers)

# Format the response as a list of dictionaries
response_json = response.json()
print(response_json[0])
```

```
{'title': 'Spotlight  ', 'year': 2015, 'awards': 2, 'nominations': 6, 'best_picture': True}
```

```python
# Format the response as a dataframe
oscars_df = pd.DataFrame(response_json)
print(oscars_df.head())
oscars_df.to_csv('data/movies_example.csv', index=False)
```

```
                 title  year  awards  nominations best_picture
0          Spotlight    2015       2            6         True
1   Mad Max: Fury Road  2015       6           10          NaN
2        The Revenant   2015       3           12          NaN
3      Bridge of Spies  2015       1            6          NaN
4        The Big Short  2015       1            5          NaN
```

```python
# Get the oscar winning films from 2010 to 2011
base_url = "https://www.scrapethissite.com/pages/ajax-javascript/"

# Initialize a list to store the dataframes
df_list = []
```

```
for year in tqdm(range(2010, 2012)):

    # Sleep for 1 to 3 seconds before each request
    sleep(random.randint(1, 3))

    # Build the url
    parameters = "?ajax=true&year=" + str(year)
    url = base_url + parameters

    # Request the page
    response = requests.get(url, headers=headers)

    # Check the status code
    if response.status_code != 200:
        print("Error with year:", year)
        continue

    # Format the response as a dictionary
    response_dict = response.json()

    # Format the response as a dataframe
    year_df = pd.DataFrame(response_dict)
    year_df["year"] = year
    df_list.append(year_df)

# Concatenate and export the dataframes
oscars_df_20102011 = pd.concat(df_list)
print(oscars_df_20102011.info())
print(oscars_df_20102011.head())
oscars_df_20102011.to_csv("data/movies_example_20102011.csv", index=False)
```

```
100%|      | 2/2 [00:04<00:00,  2.44s/it]
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28 entries, 0 to 14
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   title         28 non-null     object
 1   year          28 non-null     int64
 2   awards        28 non-null     int64
 3   nominations   28 non-null     int64
 4   best_picture  2 non-null      object
dtypes: int64(3), object(2)
memory usage: 1.3+ KB
None
                title  year  awards  nominations best_picture
0   The King's Speech  2010       4           12         True
1           Inception  2010       4            8          NaN
2  The Social Network  2010       3            8          NaN
3         The Fighter  2010       2            7          NaN
4         Toy Story 3  2010       2            5          NaN
```

### XHR-based Web Scraping: S&W

**Strengths**:

- Flexible, efficient, and scalable:
  - Can handle dynamically loaded content
  - No need to parse HTML, data is already structured (usually JSON)
- Sometimes more data available that what is shown on the webpage

**Weaknesses**:

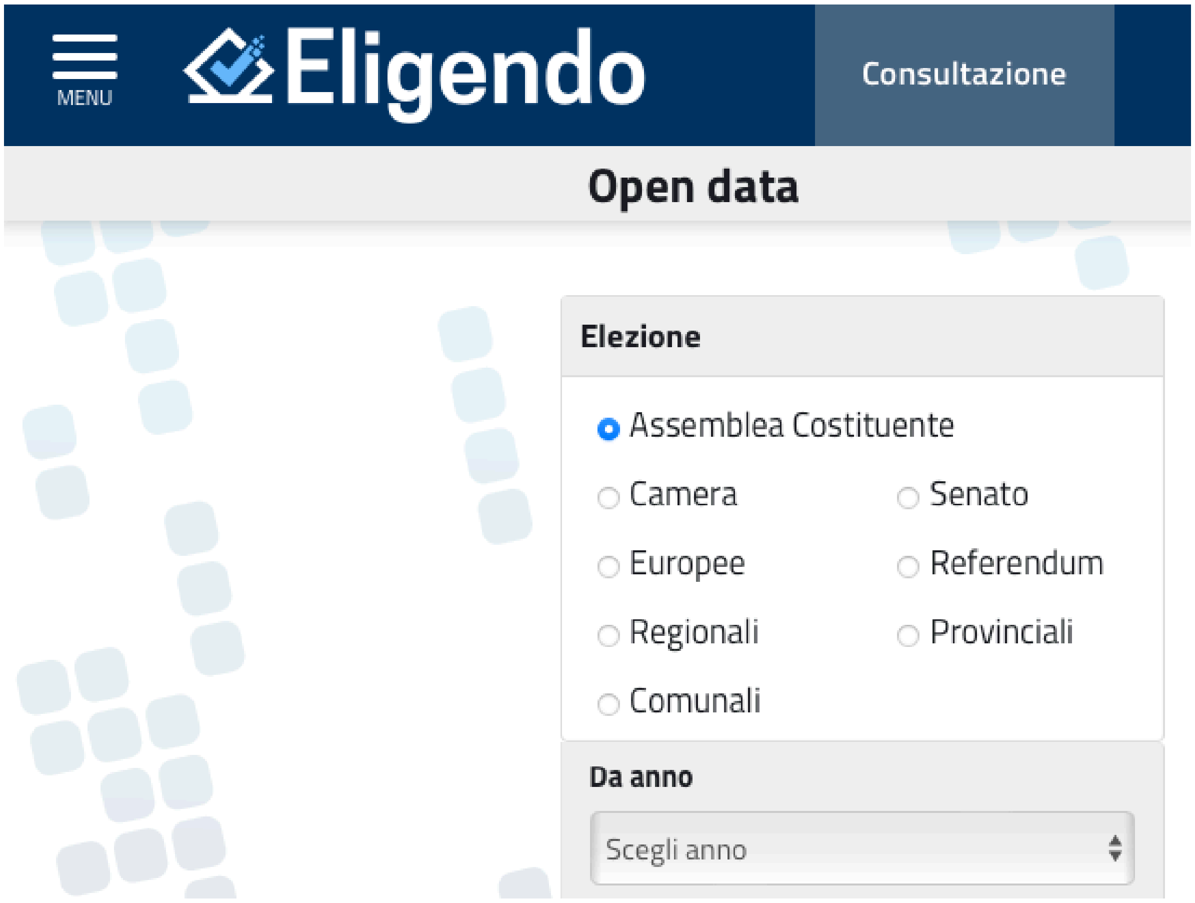
- More difficult to implement (e.g. headers, cookies, see an example here)
  - In general, requires more knowledge of the website and HTTP requests
- Not always feasible (e.g., no XHR requests, tokens required)

## 3. Automated Files Download

- In some cases, data are provided as downloadable files (e.g. csv, pdf, xls, etc.)
- It is often the case on government websites and paid databases
- Some examples: Invalsi Data, Pareri Consiglio di Stato, Top Influencers on TikTok, Spotify Charts
- However, when the number of files is large, downloading them manually can be time-consuming or unfeasible
- Furthermore, you may want to create a standardized data collection pipeline that could be shared with others

**Italian Elections Example**

Eligendo is the Open Data portal of the Italian Ministry of Interior. It contains data on the results of the Italian elections since 1946.



```python
import zipfile
import shutil

# Get the list of elections dates in Italy
url = "https://it.wikipedia.org/wiki/Calendario_delle_elezioni_in_Italia"

# Request the page
response = requests.get(url)

# Parse the page
soup = BeautifulSoup(response.content, "html.parser")
```

```python
# Get the dates
dates = [el.text for el in soup.find_all("ol")[0].find_all("li")]
print(dates[0])
```

```
18-19 aprile 1948: Elezioni politiche in Italia del 1948
```

```python
# define a function to format the dates
import locale
locale.setlocale(locale.LC_TIME, "it_IT")
from time import strptime
from datetime import datetime

def format_date(date):
    """Format election dates extracted from Wikipedia to be used in the
    Italian Elections example"""

    date_str = date.split(':')[0].split() # extract the date component
    day = date_str[0].split('-')[0]  # get the first day in case of two dates
    formatted_month = strptime(date_str[1], '%B').tm_mon  # convert the month to a number
    year = date.split()[-1]  # extract the year component from the end
    dt = datetime(year=int(year), month=int(formatted_month), day=int(day))
    formatted_date = dt.strftime('%Y%m%d')  # format the date as YYYYMMDD

    return formatted_date
```

```python
# Clean the dates with our function
formatted_dates = [format_date(date) for date in dates]
print(formatted_dates[-3:])
```

```
['20130224', '20180304', '20220925']
```

```python
# Define the base URL
base_url = "https://dait.interno.gov.it/documenti/opendata/camera/camera-"

# Create a folder to store the data
os.mkdir("data/elections")

# Download data for the last 3 elections
```

```python
for date in tqdm(formatted_dates[-3:]):

    # Sleep between 5 seconds and 10 seconds
    sleep(random.randint(5, 10))

    # Build the url
    url = base_url + date + ".zip"

    # Send the GET request (notice the importance of the headers)
    response = requests.get(url, headers=headers)

    # Check the status code
    if response.status_code != 200:
        print("Error with date:", date)
        continue

    # Store the content of the response
    with open("data/elections/camera-" + date + ".zip", "wb") as f:
        f.write(response.content)

    # Uncompress the zip file
    with zipfile.ZipFile("data/elections/camera-" + date + ".zip", "r") as zip_ref:
        zip_ref.extractall("data/elections/camera-" + date)

# Explore the data-frame
print(pd.read_csv("data/elections/camera-20220925/Camera_Italia_LivComune.txt", sep=";").h
```

```
100%|     | 3/3 [00:19<00:00,  6.35s/it]

         DATAELEZIONE CODTIPOELEZIONE    CIRC-REG         COLLPLURI  \
0  25/9/2022 00:00:00               C  PIEMONTE 1  PIEMONTE 1 - P01
1  25/9/2022 00:00:00               C  PIEMONTE 1  PIEMONTE 1 - P01
2  25/9/2022 00:00:00               C  PIEMONTE 1  PIEMONTE 1 - P01
3  25/9/2022 00:00:00               C  PIEMONTE 1  PIEMONTE 1 - P01
4  25/9/2022 00:00:00               C  PIEMONTE 1  PIEMONTE 1 - P01

                  COLLUNINOM           COMUNE  ELETTORITOT  ELETTORIM  \
0  PIEMONTE 1 - U03 (COLLEGNO)  CASELLE TORINESE        10851       5312
1  PIEMONTE 1 - U03 (COLLEGNO)  CASELLE TORINESE        10851       5312
2  PIEMONTE 1 - U03 (COLLEGNO)  CASELLE TORINESE        10851       5312
3  PIEMONTE 1 - U03 (COLLEGNO)  CASELLE TORINESE        10851       5312
```

```
4  PIEMONTE 1 - U03 (COLLEGNO)  CASELLE TORINESE        10851       5312

   VOTANTITOT  VOTANTIM  SKBIANCHE  VOTILISTA  \
0        7075      3523         84        631
1        7075      3523         84        476
2        7075      3523         84         23
3        7075      3523         84       1262
4        7075      3523         84        241

                                        DESCRLISTA   COGNOME    NOME  \
0                         LEGA PER SALVINI PREMIER  MACCANTI   ELENA
1                                     FORZA ITALIA  MACCANTI   ELENA
2         NOI MODERATI/LUPI - TOTI - BRUGNARO - UDC  MACCANTI   ELENA
3  PARTITO DEMOCRATICO - ITALIA DEMOCRATICA E PRO...  GARIGLIO  DAVIDE
4                        ALLEANZA VERDI E SINISTRA  GARIGLIO  DAVIDE

  LUOGONASCITA DATANASCITA SESSO  VOTICANDIDATO
0       TORINO  05/02/1971     F           2892
1       TORINO  05/02/1971     F           2892
2       TORINO  05/02/1971     F           2892
3       TORINO  03/04/1967     M           1853
4       TORINO  03/04/1967     M           1853
```

```python
# Delete the folder and all its content (heavy files)
shutil.rmtree("data/elections")
```

### Automated Files Download: S&W

**Strengths**:

- Depending on the source, data could be of higher quality:
  - be already structured and cleaned
  - more reliability
- Makes data collection steps reproducible
- Generally, requires fewer extractions

**Weaknesses**:

- Less control over the data collection process
- Ready-to-download files rarely available for free in many domains

## Web Scraping with Selenium

### What is Selenium?

- `selenium` is an open-source software used for automating web applications for testing purposes...
- ...but, it is also a powerful tool for scraping websites!
- It works like a robot that can click on buttons, fill out forms, and scrape data from websites.
- Useful for scraping websites where:
  - Interaction is needed (e.g., clicking on buttons, filling out forms, scrolling)
  - Content is loaded dynamically
  - Authentication is required (see below)

### Login Demo

#### NOTE

I would suggest you to use Selenium with Chrome and the code below is written for this browser. However, you can find the list of supported browsers here, and there are many tutorials online on how to use it with browsers other than Chrome.

When you run the code for the first time, please run the cell below to install the Chrome driver. See more details here

```
# from selenium import webdriver
# from selenium.webdriver.chrome.service import Service as ChromeService
# from webdriver_manager.chrome import ChromeDriverManager
# driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()

# Open the Google homepage
driver.get("https://www.google.com")
sleep(2)

# Accept the cookies
```

```
driver.find_element(By.ID, "W0wltc").click()
sleep(3)

# Search for "toscrape" in the search bar
driver.find_element(By.NAME, "q").send_keys("the internet herokuapp")
sleep(3)

# Click on the search button
driver.find_element(By.NAME, "btnK").click()
sleep(3)

# Select the first result
driver.find_element(By.XPATH ,'//*[@id="rso"]/div[1]/div/div/div/div/div/div/div/div[1]/di
sleep(3)

# Scroll down and select the "Form Authentication" link
driver.execute_script("window.scrollTo(0, 500)")
sleep(3)
driver.find_element(By.XPATH, '//*[@id="content"]/ul/li[21]/a').click()

# As you can see, you can search for elements using different methods
# Insert username and password (search by ID and NAME)
driver.find_element(By.ID, "username").send_keys('tomsmith')
sleep(2)
driver.find_element(By.NAME, "password").send_keys('SuperSecretPassword!')
sleep(2)
# Submit (search by TAG_NAME)
driver.find_element(By.TAG_NAME, "button").click()
sleep(4)

# Logout (search by XPATH)
driver.find_element(By.XPATH, '//*[@id="content"]/div/a/i').click()
sleep(3)

# Close the browser
driver.quit()
```

**Selenium: S&W**

**Strengths:**

- Flexible and powerful framework:
  - Can handle dynamically loaded content
  - Can interact with the webpage and perform complex actions

**Weaknesses**:

- More difficult to implement and maintain
- Slow and resource-intensive → limited scalability

## How to Approach Web Scraping

### Be Responsible

- Try to find an official API first or check if data is already available for download (e.g. on Kaggle, Dataverse)
- Review the website's Terms of Service and robots.txt file to check if web scraping is allowed
- Comply with privacy and data protection policies when dealing with personal or sensitive data
- Set a reasonable scraping rate to avoid overloading the website's server

### Be Smart

- Is web scraping the best solution to your problem?
- Check if other people have already scraped the website you are interested in.
- Always evaluate the quality and the representativeness of the data you are scraping.
- Test your code on a small sample of data before scaling up.

### Further Readings

- Applications of Web Scraping in Economics and Finance (2022)
- Fields of Gold: Scraping Web Data for Marketing Insights (2022)
- Using Internet Data for Economic Research (2012)