What is Web Scraping?

Web scraping **is an automatic method to obtain large amounts of data from websites.** Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications

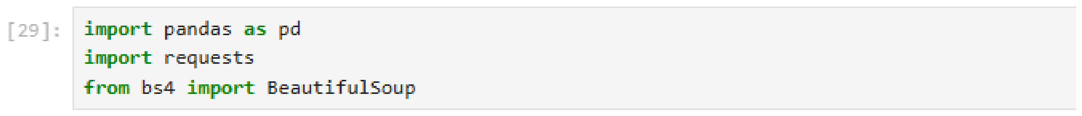
There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API's or even creating your code for web scraping from scratch. **Many large websites, like Google, Twitter, Facebook, StackOverflow, etc. have API's that allow you to access their data in a structured format.** This is the best option, but there are other sites that don't allow users to access large amounts of data in a structured form or they are simply not that technologically advanced. In that situation, it's best to use Web Scraping to scrape the website for data.

Web scraping requires two parts, namely the **crawler** and the **scraper**. **The crawler is an artificial intelligence algorithm that browses the web to search for the particular data required by following the links across the internet. The scraper, on the other hand, is a specific tool created to extract data from the website.** The design of the scraper can vary greatly according to the complexity and scope of the project so that it can quickly and accurately extract the data.

🕸 Web Scraping - Overview

- **Goal**: Extract company details like name, rating, reviews, location, tags, and about section from a web page.
- **Problem Faced**: Access to some pages was denied due to server-side protection (403 Forbidden error).

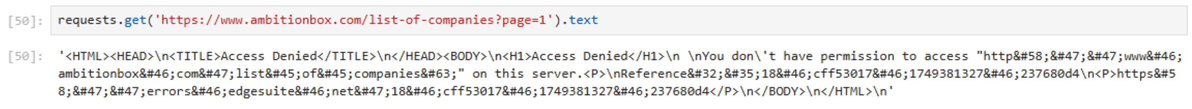
🔧 Libraries Required



```
[29]: import pandas as pd
      import requests
      from bs4 import BeautifulSoup
```

- These libraries are available by default in **Anaconda**, so no need for separate installation.

🚫 403 Error - Access Denied

- Websites can block automated scrapers using the robots.txt file.
- Direct requests using requests.get() may be rejected.
- 

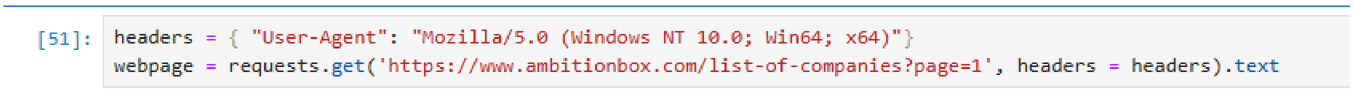
```
[50]: requests.get('https://www.ambitionbox.com/list-of-companies?page=1').text

[50]: '<HTML><HEAD>\n<TITLE>Access Denied</TITLE>\n</HEAD><BODY>\n<H1>Access Denied</H1>\n \nYou don\'t have permission to access "http&#58;&#47;&#47;www&#46;
      ambitionbox&#46;com&#47;list&#45;of&#45;companies&#63;" on this server.<P>\nReference&#32;&#35;18&#46;cff53017&#46;1749381327&#46;237680d4\n<P>https&#5
      8;&#47;&#47;errors&#46;edgesuite&#46;net&#47;18&#46;cff53017&#46;1749381327&#46;237680d4</P>\n</BODY>\n</HTML>\n'
```

Solution:

Use **User-Agent headers** to mimic a browser request.

python

```
headers = {
  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}
response = requests.get(url, headers=headers)
```



```
[51]: headers = { "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
      webpage = requests.get('https://www.ambitionbox.com/list-of-companies?page=1', headers = headers).text
```

- This tricks the server into thinking the request is coming from a real browser.

🌐 Viewing Page Source vs Inspect Tool

- **View Page Source**: Shows the static HTML served by the server.
- **Inspect Element**: Helps identify specific HTML tags related to required data.

Use this to locate:

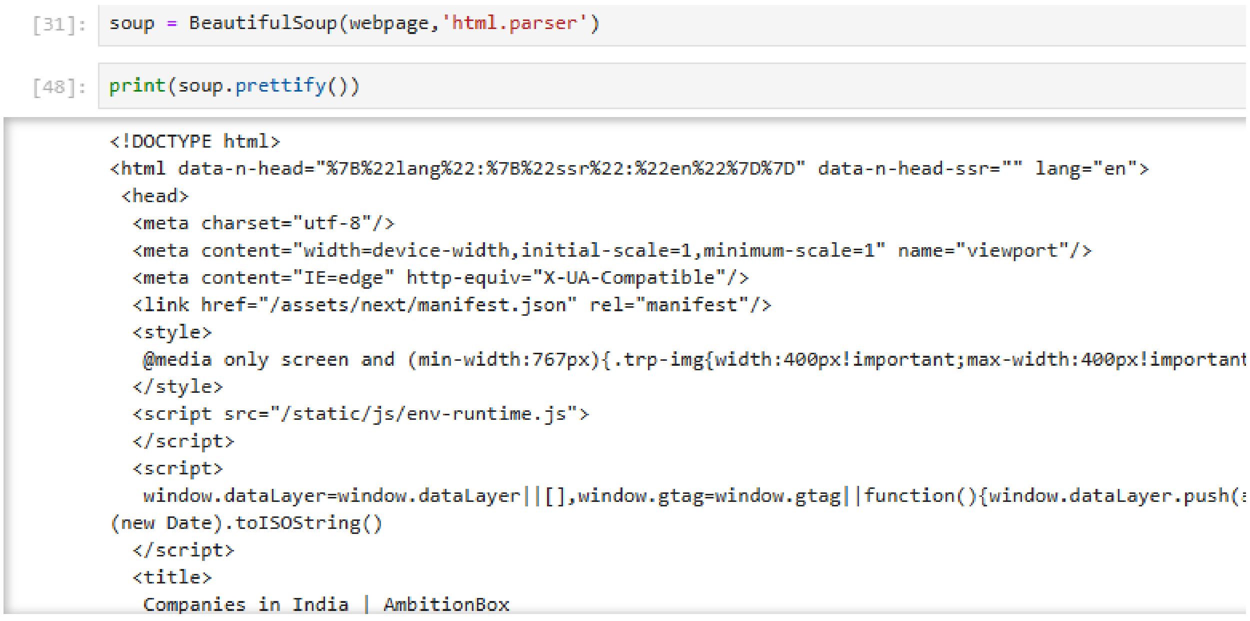
- Company container tags
- Class or ID of the section (e.g., <div class="company-block">)

🧪 Parsing the HTML

Python

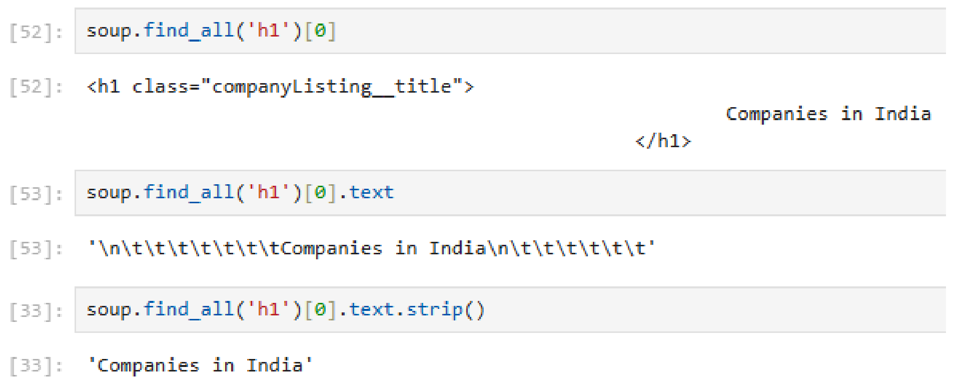
soup = BeautifulSoup(response.content, 'html.parser')

- Parses the HTML content into a navigable BeautifulSoup object.
- You can now use .find() or .find_all() to extract specific elements.



```
[31]: soup = BeautifulSoup(webpage,'html.parser')

[48]: print(soup.prettify())

<!DOCTYPE html>
<html data-n-head="%7B%22lang%22:%7B%22ssr%22:%22en%22%7D%7D" data-n-head-ssr="" lang="en">
 <head>
  <meta charset="utf-8"/>
  <meta content="width=device-width,initial-scale=1,minimum-scale=1" name="viewport"/>
  <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
  <link href="/assets/next/manifest.json" rel="manifest"/>
  <style>
   @media only screen and (min-width:767px){.trp-img{width:400px!important;max-width:400px!important
  </style>
  <script src="/static/js/env-runtime.js">
  </script>
  <script>
   window.dataLayer=window.dataLayer||[],window.gtag=window.gtag||function(){window.dataLayer.push(
(new Date).toISOString()
  </script>
  <title>
   Companies in India | AmbitionBox
```

🖼️ Extracting Data Example



```
[52]: soup.find_all('h1')[0]

[52]: <h1 class="companyListing__title">
                                                        Companies in India
                                                </h1>

[53]: soup.find_all('h1')[0].text

[53]: '\n\t\t\t\t\t\t\tCompanies in India\n\t\t\t\t\t\t'

[33]: soup.find_all('h1')[0].text.strip()

[33]: 'Companies in India'
```

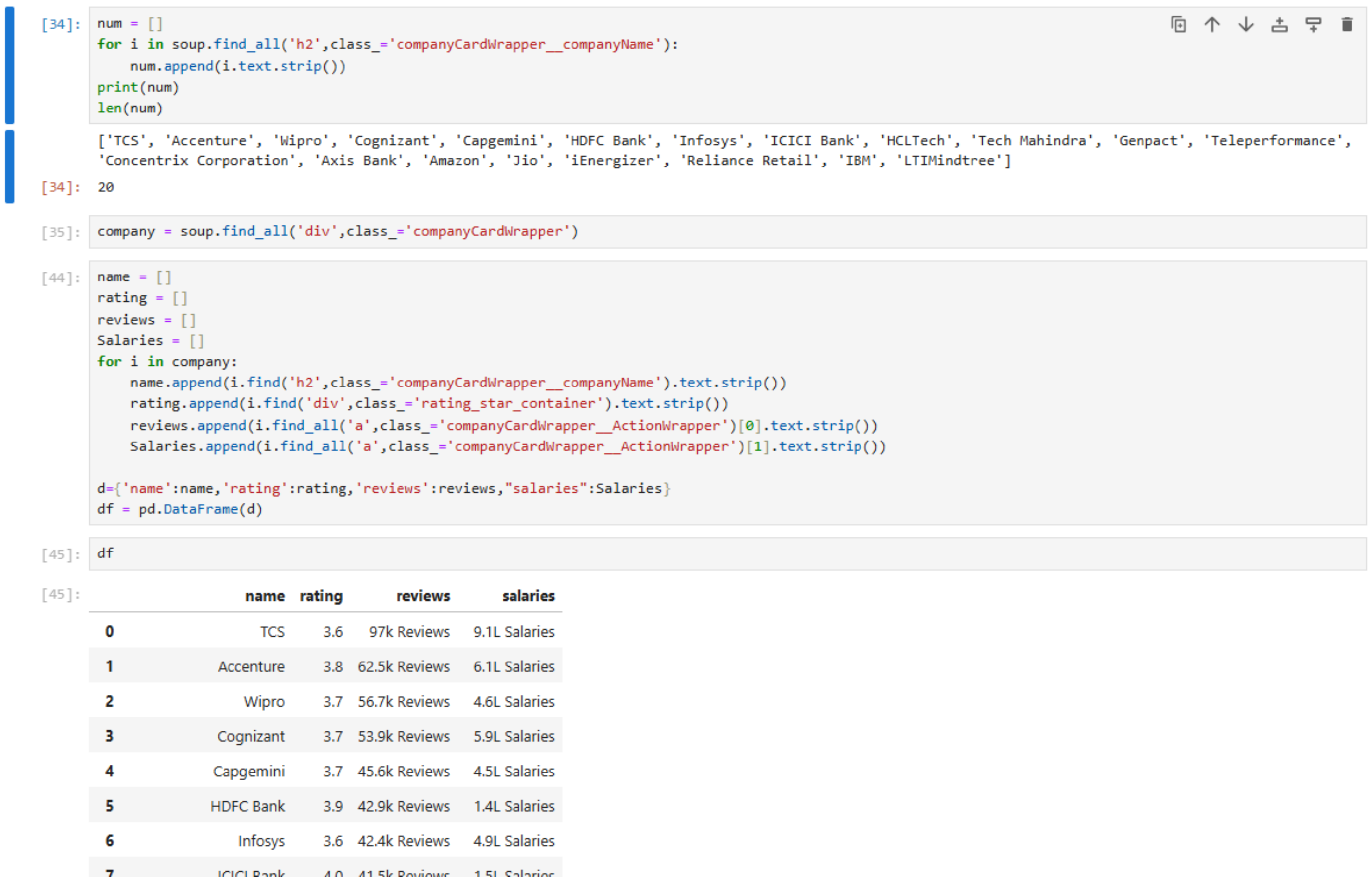
Assuming you want to extract:

- Company Name
- Rating
- Reviews
- Headquarters
- Tags
- About section

Example code:

python

```
companies = soup.find_all('div', class_='company-block')  # hypothetical class

for company in companies:
    name = company.find('h3').text.strip()
    rating = company.find('span', class_='rating').text.strip()
    reviews = company.find('span', class_='reviews').text.strip()
    location = company.find('span', class_='location').text.strip()
    tags = ", ".join([tag.text for tag in company.find_all('span', class_='tag')])
    about = company.find('p', class_='about').text.strip()

print(name, rating, reviews, location, tags, about)
```



```
[34]: num = []
      for i in soup.find_all('h2',class_='companyCardWrapper__companyName'):
          num.append(i.text.strip())
      print(num)
      len(num)

      ['TCS', 'Accenture', 'Wipro', 'Cognizant', 'Capgemini', 'HDFC Bank', 'Infosys', 'ICICI Bank', 'HCLTech', 'Tech Mahindra', 'Genpact', 'Teleperformance',
      'Concentrix Corporation', 'Axis Bank', 'Amazon', 'Jio', 'iEnergizer', 'Reliance Retail', 'IBM', 'LTIMindtree']
[34]: 20

[35]: company = soup.find_all('div',class_='companyCardWrapper')

[44]: name = []
      rating = []
      reviews = []
      Salaries = []
      for i in company:
          name.append(i.find('h2',class_='companyCardWrapper__companyName').text.strip())
          rating.append(i.find('div',class_='rating_star_container').text.strip())
          reviews.append(i.find_all('a',class_='companyCardWrapper__ActionWrapper')[0].text.strip())
          Salaries.append(i.find_all('a',class_='companyCardWrapper__ActionWrapper')[1].text.strip())

      d={'name':name,'rating':rating,'reviews':reviews,"salaries":Salaries}
      df = pd.DataFrame(d)

[45]: df

[45]:
```

	name	rating	reviews	salaries
0	TCS	3.6	97k Reviews	9.1L Salaries
1	Accenture	3.8	62.5k Reviews	6.1L Salaries
2	Wipro	3.7	56.7k Reviews	4.6L Salaries
3	Cognizant	3.7	53.9k Reviews	5.9L Salaries
4	Capgemini	3.7	45.6k Reviews	4.5L Salaries
5	HDFC Bank	3.9	42.9k Reviews	1.4L Salaries
6	Infosys	3.6	42.4k Reviews	4.9L Salaries
7	ICICI Bank	4.0	41.5k Reviews	1.5L Salaries

🔍 Using Inspect Tool for Tag Selection

Steps:

1. Open browser → Right-click → **Inspect**
2. Click the **element picker tool**.
3. Hover over the section (e.g., "TCS").
4. Observe which tag and class is responsible.
5. Use those tags/classes in your BeautifulSoup code.

🧹 Cleaning & Formatting HTML

Use soup.prettify() to print a readable HTML structure:

python

Copy code

print(soup.prettify())

Useful for debugging and understanding the hierarchy.

✅ Summary Steps for Web Scraping

1. Import required libraries.

2. Use requests with proper headers to fetch page content.
3. Parse HTML with BeautifulSoup.
4. Inspect web page to identify correct tags/classes.
5. Extract and clean data using .find()/.find_all().
6. Format or save data as needed (CSV, JSON, etc.).

📝 Notes: Web Scraping Using BeautifulSoup – Targeting Company Data

🔍 1. Using Inspect Element for Data Identification

- Use browser **Inspect Element** to highlight the exact HTML section where the data (e.g., TCS) is located.
- On hover, the browser highlights the **parent container (div, section, etc.)** containing that element.
- Identifying the **repeated structure** (e.g., multiple company blocks) is crucial for scraping multiple entries.

🛠 2. Understanding the Repeating Pattern

- Example: TCS, Accenture, ICICI Bank all appear in similar containers.
- These containers have **identical tag structures and classes**.
- Use this pattern to apply soup.find_all() and fetch all such blocks.

🧪 3. Basic BeautifulSoup Usage

python

Copy code

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(webpage, 'lxml')  # use 'html.parser' if lxml is unavailable
```

🔄 4. Extracting Repeated Elements

python

Copy code

```
containers = soup.find_all('div', class_='company-container')  # adjust tag/class
```

- Each container holds information like:
  - Company name
  - Ratings
  - Other metadata

🔠 5. Extracting Company Names

python

```
for company in containers:
  name = company.find('h2').text.strip()
  print(name)
```

- .find('h2'): Finds the name tag
- .text.strip(): Removes whitespace or newline characters

🏷️ 6. Extracting Additional Info (e.g., Ratings)

python

```
  rating = company.find('span', class_='rating').text.strip()
  print(f"Company: {name}, Rating: {rating}")
```

- You can extract multiple details (like location, category) from the same container.

🧼 7. Cleaning Special Characters / HTML Formatting

- Sometimes extracted data has **special characters** like non-breaking spaces (\xa0) or unicode symbols.
- Use .replace() or regex to clean:

python

```
clean_text = raw_text.replace('\xa0', ' ')
```

📌 8. Edge Cases

- Sometimes hidden characters or white space may not show up visibly.
- Use debugging (print(repr(text))) to verify hidden formatting issues.

💡 Bonus Tip

- Use len(soup.find_all('h2')) to check how many items are being captured.
- Helps in validating if you're scraping the correct section.