




WeCloudData

Web Scrapping

Prepared by WeCloudData



What is Web Scraping?

Why scrape?

- **Introduction to Web Scraping**
- Understanding HTML
- Tools and Libraries Overview
- Advanced Techniques using Scrapy
- Handling Data Post-Scraping

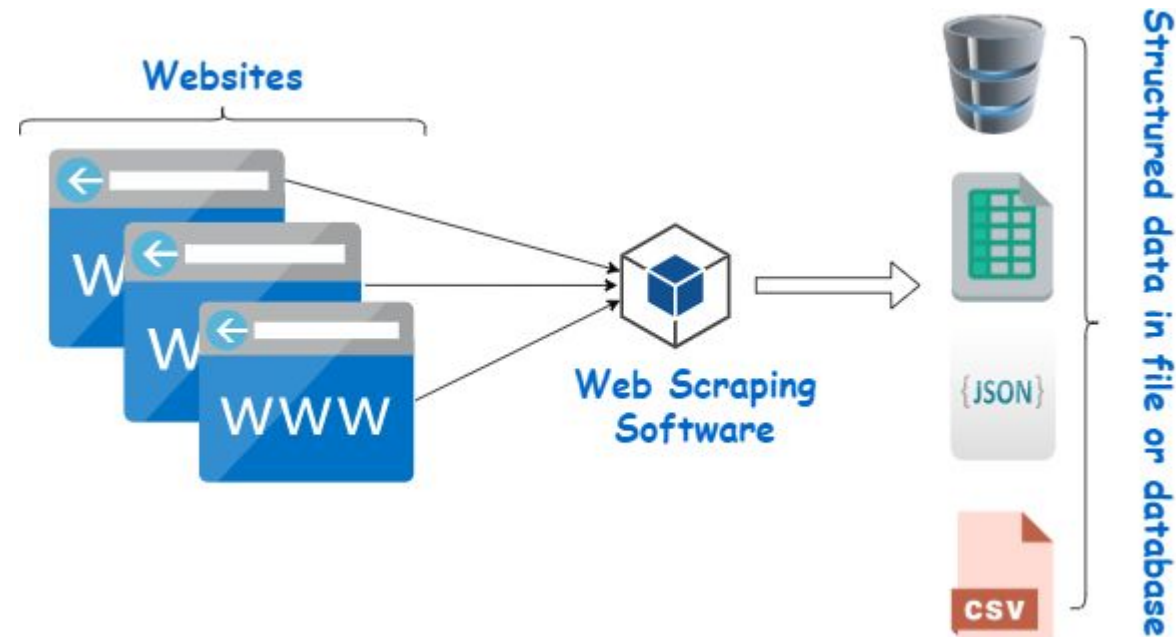
Agenda.



Introduction to Web Scraping

What is it?

- Web scraping, also known as web harvesting or web data extraction, is the process of extracting data from websites.
- This is done through the use of software that simulates human web surfing to collect specified bits of information from different websites.





Introduction to Web Scraping

Why scrape?

- Why is Web Scraping Important?
 - Web scraping is essential for gathering large amounts of data from the internet. This data can then be analyzed, compared, and processed for various insights and applications, such as competitive analysis, market research, financial insights, etc.
- Individuals and companies might use web scraping to:
 - Gather product listings from e-commerce sites.
 - Monitor data from real estate marketplaces.
 - Aggregate news articles or other forms of content.
 - Extract data for machine learning models.
 - Compile public data sets (like job postings or event listings).

- Introduction to Web Scraping
- **Understanding HTML**
- Tools and Libraries Overview
- Advanced Techniques using Scrapy
- Handling Data Post-Scraping

Agenda.



Understanding HTML

HTML Basics

- In order to learn about web scraping, we must first learn about HTML.
- HTML, or HyperText Markup Language, is the standard language for creating web pages.
- It structures the web content and is fundamental for web scraping.
- Understanding HTML is key to effectively extract information from websites.



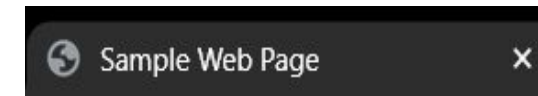


Understanding HTML

The Structure of an HTML Document

- Every HTML document follows a basic structure.
- The `<!DOCTYPE html>` declaration defines the document type.
- The `<html>` element wraps the entire content.
- Inside `<html>`, `<head>` contains meta-information, and `<body>` contains the webpage's visible content.
- Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Web Page</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="A simple HTML page">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a paragraph of text in the body of the webpage.</p>
</body>
</html>
```



Welcome to My Web Page

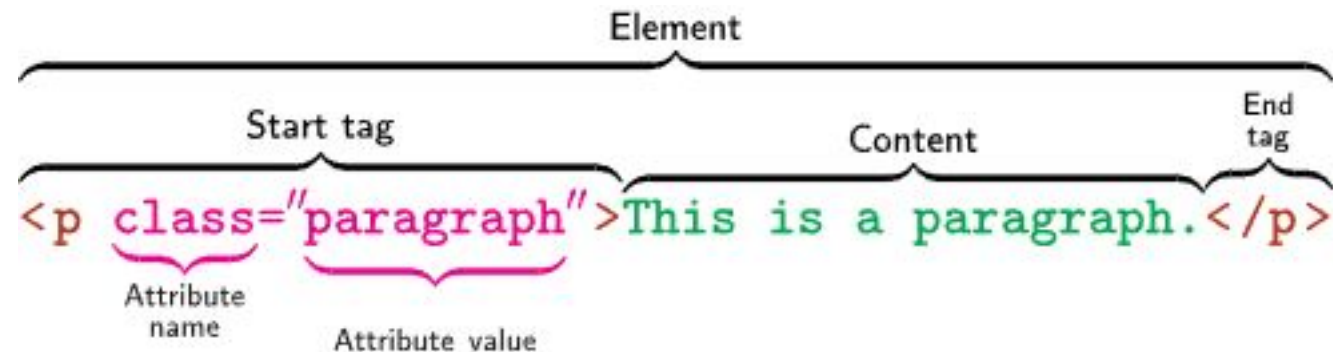
This is a paragraph of text in the body of the webpage.



Understanding HTML

Understanding HTML Tags

- HTML Tags are the basic building blocks of a webpage.
- A tag represents different content types like headings, paragraphs, links, etc.
- Tags usually come in pairs: an opening tag `<tag>` and a closing tag `</tag>`.
- Example: `<p>` is a paragraph tag used to define a text block.





Understanding HTML

HTML Elements

- An HTML Element includes the opening tag, the closing tag, and the content in between.
- Elements can contain plain text, other elements, or both, forming a nested structure.
- Example: `<p>This is a paragraph.</p>` is a paragraph element.

```
<!Doctype html>
<html>
  <head>
    <title>
      This is the title
    </title>
  </head>
  <body>
    <div>
      <p>Hello world!</p>
    </div>
  </body>
</html>
```

Hello world!



Understanding HTML

HTML Attributes

- Attributes in HTML provide additional information about elements.
- They are defined within the opening tag in a name/value pair format.
- Example:
 - In `Visit Example!`, href is an attribute specifying the link's destination.

```
<div>
  <h1>Slide Title</h1>
  <p>This is the content of the slide.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
  <a href="https://www.example.com">Visit Example!</a>
```

Slide Title

This is the content of the slide.

- Item 1
- Item 2
- Item 3

[Visit Example!](https://www.example.com)



Understanding HTML

Nested HTML Elements

- "Elements can be nested inside other elements, creating a hierarchical structure."
- "This nesting is crucial for organizing and grouping content on a webpage."
- "Example: A <div> element can contain multiple <p> elements."

```
<!Doctype html>
<html>
  <head>
    <title>
      This is the title
    </title>
  </head>
  <body>
    <div>
      <p>Hello world!</p>
      <p>Welcome to this lecture!</p>
      <p>Today we will learn about HTML!</p>
    </div>
  </body>
```

Hello world!

Welcome to this lecture!

Today we will learn about HTML!



Understanding HTML

Understanding XPATH

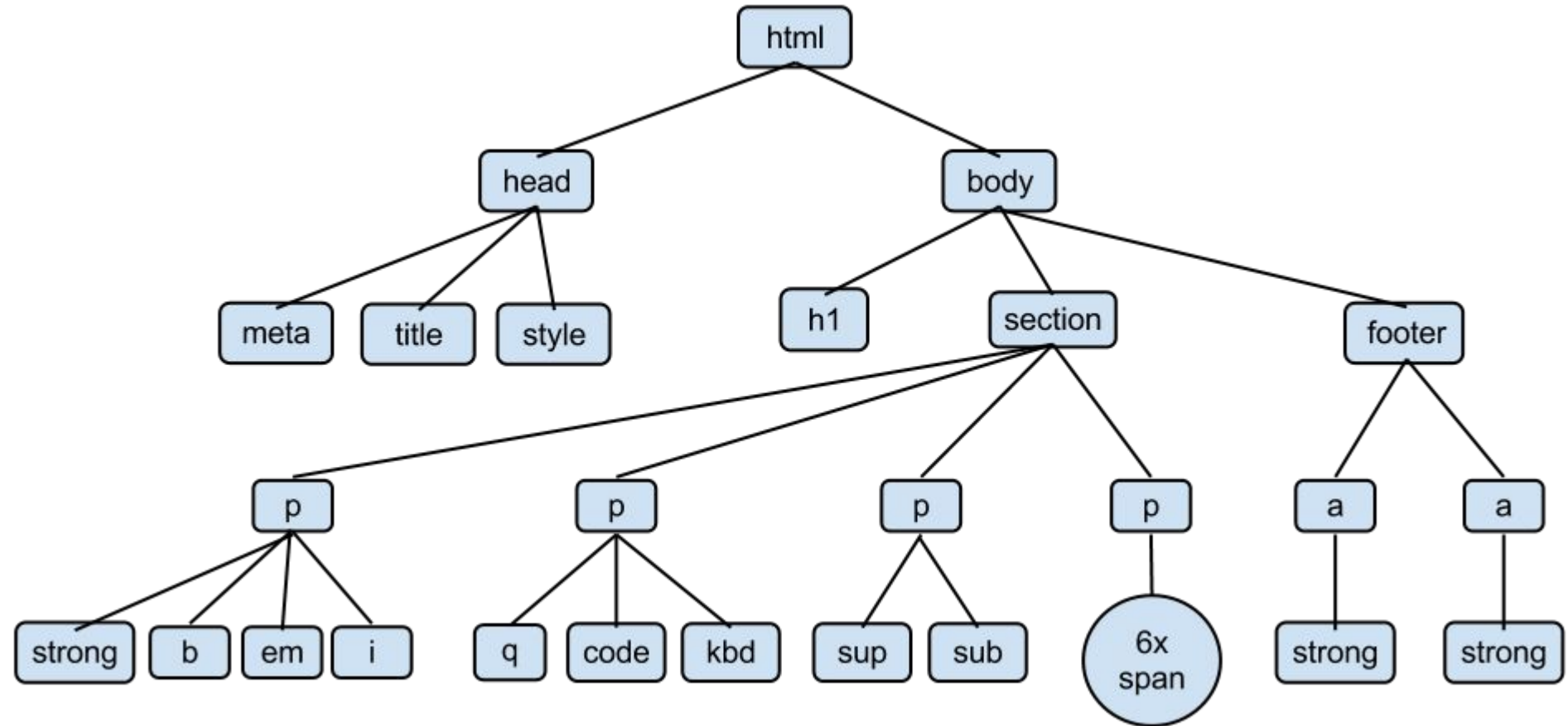
- XPath stands for XML Path Language. It's a syntax used for selecting nodes in XML or HTML documents.
- XPath expressions allow for navigation in the structure, with the ability to find elements by attributes, node names, or within the document hierarchy.
- Useful in web scraping for pinpointing specific parts of a webpage, especially when elements lack unique IDs or classes.
- Syntax Example: `//tagname[@attribute='value']` selects all nodes with the specified tag, attribute, and value.



Understanding HTML

Understanding XPATH

HTML Tree Structure





Understanding HTML

QUERY

```
/html/body/div/div[1]/p[2]/text()
```

RESULTS (1)

In the past 6 months, I have completed 3 data engineering projects, covering sql, python, machine learning, and big data.

XPATH

by WeCloudData Academy



This web page contains the data engineering projects I worked

#text 535.95 × 45 journey.

In the past 6 months, I have completed 3 data engineering projects, covering sql, python, machine learning, and big data.

Location in HTML

- Job market analysis with Pandas and Beautifulsoup ([link](#))
- Customer churn analysis with predictive modeling ([link](#))
- Sentiment classification with deep learning ([link](#))
- Big data analysis with spark and AWS ([link](#))

Project Title	Tools Used
Job market analysis with Pandas and Beautifulsoup	Pandas, Python, Beautisoup, Selenium
Customer churn analysis with predictive modeling	Python, Scikit-learn, Machine Learning, Classification
Sentiment classification with deep learning	Python, NLTK, PyTorch



Elements

Content

```

<div class="row">
  <div class="column1">
    <p>...</p>
    ...
  </div>
  <div class="column2">...</div>
</div>
<table>
  <tbody>
    <tr>
      <th> Project Title </th>
      <th> Tools Used </th>
    </tr>
    <tr>...</tr>
    <tr>...</tr>
    <tr>...</tr>
  </tbody>
</table>
<div class="column2">...</div>
::after
</div>
<iframe src="chrome-extension://hgimnogjllphhhkhlmebbm_lgjoejdpjl/bar.html" id="xh-bar" class>...</iframe>
</body>
<style> @media print { #simplifyJobsContainer { display: none; } } </style>
<div id="simplifyJobsContainer" style="position: absolute; top: 0px; left: 0px; width: 0px; height: 0px; overflow: visible; z-index: 2147483647;">...</div>
<script id="simplifyJobsPageScript" src="chrome-extensio
html body div.row div.column1 p (text)
```

WeCloudData





Understanding HTML

Understanding CSS

- CSS Selectors are patterns used to select the elements you want to style in a web page. In web scraping, they are used to locate and extract data from specific parts of the HTML document.
- Selectors can be based on element names, classes, IDs, attributes, and more.
- Simple and intuitive, making them popular for web scraping tasks.
- Syntax Example: `.classname` selects all elements with the specified class, and `#idname` selects the element with the specified ID.



Understanding HTML

Understanding CSS

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <div id="header">
    <h1 class="title">Welcome to My Page</h1>
  </div>
  <div class="content">
    <p>This is a sample paragraph in the content area.</p>
    <a href="#" class="link">Click Here</a>
  </div>
  <ul class="menu">
    <li>Home</li>
    <li>About</li>
    <li>Contact</li>
  </ul>
</body>
</html>
```

CSS

```
#header {
  background-color: lightblue;
  padding: 10px;
  text-align: center;
}

.title {
  color: darkblue;
}

.content p {
  font-size: 16px;
}

.link {
  color: green;
  text-decoration: none;
}

.menu {
  list-style-type: none;
  padding: 0;
}

.menu li {
  display: inline;
  margin-right: 10px;
}
```



Understanding HTML

Understanding CSS

Welcome to My Page

This is a sample paragraph in the content area.

[Click Here](#)

[Home](#) [About](#) [Contact](#)

Demo

Web-scraping-html-bs4-demo.ipynb Section 1 ([LINK](#))



WeCloudData

📍 500-80 Bloor Street West, Toronto

📍 ON

www.weclouddata.com



- Introduction to Web Scraping
- Understanding HTML
- **Tools and Libraries Overview**
- Advanced Techniques using Scrapy
- Handling Data Post-Scraping

Agenda.



Tools and Libraries Overview

Basic Web Scraping with `requests` and `BeautifulSoup`

- Explore the fundamentals of web scraping with two Python libraries: requests and BeautifulSoup.
- Ideal for basic scraping tasks, offering a simple introduction to data extraction.



BeautifulSoup



Tools and Libraries Overview

Basic Web Scraping with `requests` and `BeautifulSoup`

- requests: A user-friendly HTTP library for sending HTTP/1.1 requests simply.
 - It fetches HTML/XML data
- BeautifulSoup: A library for parsing HTML and XML documents, enabling easy navigation and search through the parse tree for web scraping.
 - BeautifulSoup parses HTML/XML data.



Tools and Libraries Overview

Installation of requests and BeautifulSoup

- In order to have requests and BeautifulSoup running in your Jupyter Notebook, you would need to install them using the following commands:

```
%pip install requests  
%pip install beautifulsoup4
```



Tools and Libraries Overview

Basic Web Scraping Process

- **Send an HTTP Request:** Use `requests.get()` to access the target website.
 - The first step in web scraping is to access the target website.
 - We use the `requests.get()` method to make a GET request to the website.

```
import requests
URL = 'https://www.example.com'
page = requests.get(URL)
```

- **Parse the Content:** Utilize BeautifulSoup to parse the fetched webpage content.
 - After fetching the webpage content, we need a way to parse and extract the information we need. This is where `BeautifulSoup` comes into play.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
```




Tools and Libraries Overview

Basic Web Scraping Process

- **Extract Data:** Navigate the parse tree to find and extract the desired data.
 - Once you've created a `BeautifulSoup` object, you can find specific data by navigating through the parse tree. You can search for data using tag names, IDs, classes, and much more.

```
# Find the first element with the 'class' attribute set to 'example'
example_element = soup.find(class_='example')

# Find all instances of a tag
for element in soup.find_all('a'):
    print(element.get('href'))
```

- **Output the Data:** Save the extracted data to a file or database.
 - After extracting the desired data, you might want to save it in a file or database for further processing or analysis.

```
with open('output.txt', 'w') as file:
    file.write(str(example_element))
```

Demo

Web-scraping-html-bs4-demo.ipynb Section 2 ([LINK](#))



WeCloudData

📍 500-80 Bloor Street West, Toronto

📍 ON

www.weclouddata.com



- Introduction to Web Scraping
- Understanding HTML
- Tools and Libraries Overview
- **Advanced Techniques using Scrapy**
- Handling Data Post-Scraping

Agenda.



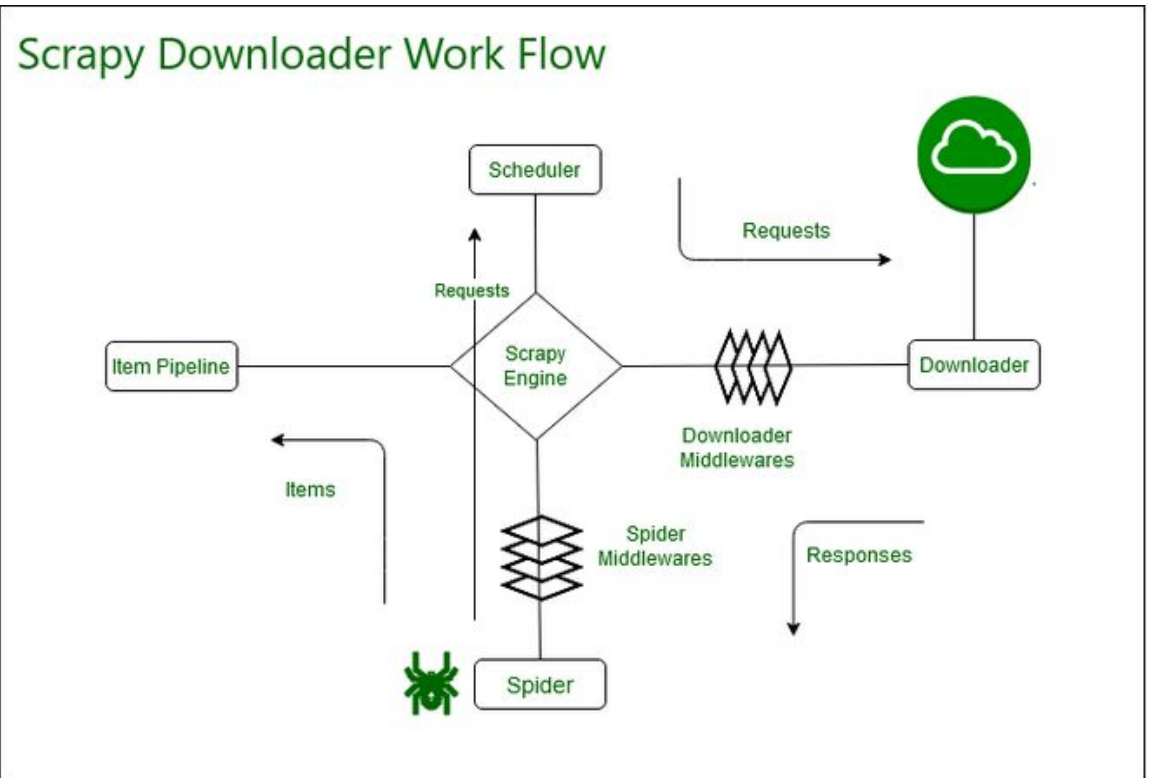
Advanced Techniques using Scrapy

Introducing Scrapy

- Scrapy: A powerful open-source web-crawling and web-scraping framework written in Python.
- Designed for efficient data extraction from websites and large-scale web scraping.



Scrapy





Advanced Techniques using Scrapy

Why Scrapy?

- There are several reasons to choose Scrapy for your web scraping needs:
 - **Robustness:** Scrapy is well-suited for large-scale and complex web scraping tasks. It's designed to handle requests asynchronously for speed, and includes several built-in features for handling errors and retrying requests.
 - **Speed:** Due to its asynchronous handling of requests, Scrapy is fast. It can handle multiple requests in parallel, significantly reducing the time required to scrape pages.
 - **Extensibility:** Scrapy is highly extensible, allowing you to plug in new functionality easily through middlewares, extensions, and pipelines.
 - **Ease of Use:** Despite its sophistication, Scrapy provides a convenient command-line interface to streamline the creation of new projects and spiders, making it relatively easy to use once you understand the basics.
 - **Comprehensive:** Scrapy provides everything you need in one package, from making requests and selecting data, to handling data pipelines, stats collection, caching, and more.



Advanced Techniques using Scrapy

Scrapy Architecture

- Brief overview of Scrapy's architecture:
- **Engine**
 - The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur.
- **Scheduler**
 - The scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.
- **Downloader**
 - The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.
- **Spiders**
 - Spiders are custom classes written by Scrapy users to parse responses and extract items from them or additional requests to follow.
- **Item Pipeline**
 - The Item Pipeline is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database).



Advanced Techniques using Scrapy

Installation of Scrapy

- It is recommended to install Scrapy inside a virtual environment.
- This can be done using the following command for Ubuntu:

```
pip install scrapy
```

- To follow the steps required for Windows / macOS / Linux (Ubuntu), please follow the guide from the Scrapy documentation website:
<https://docs.scrapy.org/en/latest/intro/install.html>



Advanced Techniques using Scrapy

Starting with Scrapy

- To create a new Scrapy project using `myproject` as your project name:

```
scrapy startproject myproject
```

- Scrapy will automatically create the project folder and the following files:

```
> spiders
+ __init__.py
+ items.py
+ middlewares.py
+ pipelines.py
(i) README.md
≡ requirements.txt
+ settings.py
```


- Introduction to Web Scraping
- Understanding HTML
- Tools and Libraries Overview
- Advanced Techniques using Scrapy
- **Handling Data Post-Scraping**

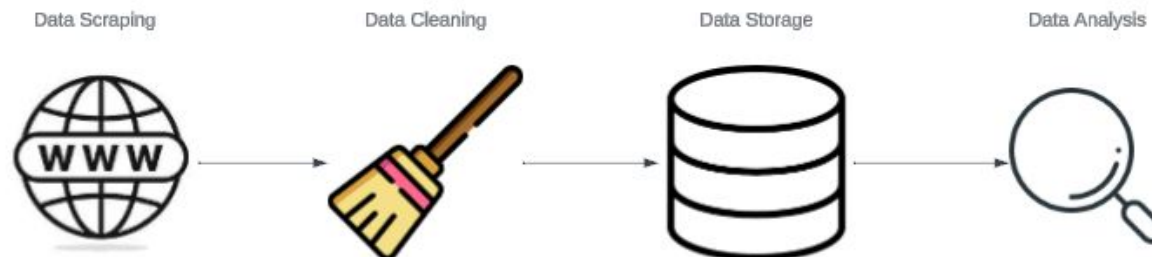
Agenda.



Handling Data Post-Scraping

What to Do After Data Scraping?

- **Data Scraping:** Just the Beginning
 - Data scraping is the first step in a broader data processing workflow. What follows is equally crucial for deriving value from the scraped data.
- **Understanding Post-Scraping Workflow**
 - Post-scraping involves several key steps: Data cleaning, data storage, and data analysis.
 - Each step is vital to ensure the usability and integrity of the data collected.
- **Data Cleaning:** Ensuring Quality and Accuracy
 - Scraped data often comes with inconsistencies, errors, or irrelevant information.
 - Data cleaning includes tasks like removing duplicates, correcting errors, standardizing formats, and filtering out noise.

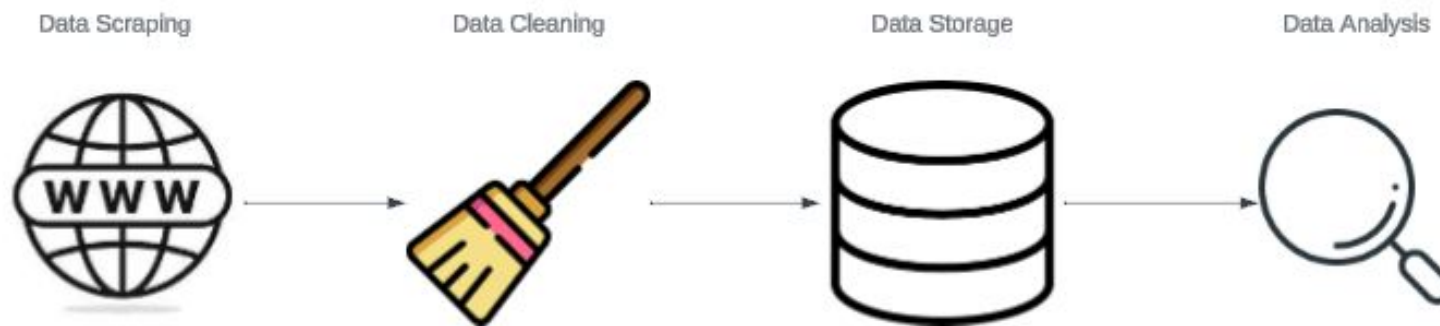




Handling Data Post-Scraping

What to Do After Data Scraping?

- **Data Storage:** Securing Your Data
 - Once cleaned, data needs to be stored in a structured format that facilitates easy access and analysis.
 - Options range from simple file systems (CSV, JSON files) to more complex database solutions (SQL, NoSQL databases).
- **Data Analysis:** Extracting Insights
 - The ultimate goal of scraping data is to analyze it for insights.
 - Data analysis involves examining, transforming, and modeling data to discover useful information, answer questions, or support decision-making.





Handling Data Post-Scraping

Cleaning and Processing Scraped Data

- Scraped data often includes unwanted noise, errors, and inconsistencies.
- Data cleaning involves removing duplicates, correcting errors, and standardizing data formats.
- Processing may include transforming data types, normalizing text, and aggregating information.
- Effective cleaning ensures the accuracy and usefulness of data for analysis.

(a) Dirty Data

id	title	pub_year	citation_count
t1	CrowdDB	11	18
t2	TinyDB	2005	1569
t3	YFilter	Feb, 2002	298
t4	Aqua		106
t5	DataSpace	2008	107
t6	CrowdER	2012	1
t7	Online Aggr.	1997	687
...
t10000	YFilter - ICDE	2002	298

(b) Cleaned Sample

id	title	pub_year	citation_count	#dup
t1	CrowdDB	2011	144	2
t2	TinyDB	2005	1569	1
t3	YFilter	2002	298	2
t4	Aqua	1999	106	1
t5	DataSpace	2008	107	1
t6	CrowdER	2012	34	1
t7	Online Aggr.	1997	687	3



Handling Data Post-Scraping

Storing Your Scraped Data

- Proper storage is crucial for large-scale data analysis and retrieval.
 - Options include flat files (CSV, JSON), databases (SQL, NoSQL), and cloud storage.
 - Choice depends on data size, complexity, and intended use.
 - Organize data in a structured, query-able format for easy access.
-
- Types of storage:



Local storage



Database



Cloud storage



Handling Data Post-Scraping

Making the Most of Your Data

- Data analysis involves extracting insights and patterns from your dataset.
- Utilize tools like Python's Pandas, R, or BI software for analysis and visualization.
- Data can be used for reporting, decision-making, machine learning models, and more.
- Focus on converting raw data into actionable insights.



Programming Languages



Power BI



+tableau

BI Software



Handling Data Post-Scraping

Best Practices in Data Handling

- Always consider the legal implications of web scraping.
- Adhere to a website's robots.txt guidelines and terms of service.
- Implement rate limiting to avoid overloading servers.
- Respect user privacy and data protection laws (like GDPR - General Data Protection Regulation).
- Transparency and ethical practices ensure long-term sustainability of web scraping projects.

← → ↻ google.com/robots.txt 🔍

- Examples of robots.txt:

```
User-agent: *
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
Disallow: /?hl=*
Allow: /?hl=*gws_rd=ssl$
Disallow: /?hl=*gws_rd=ssl$
```

Thank You!



WeCloudData

📍 500-80 Bloor Street West, Toronto

📍 ON

www.weclouddata.com

