Data Science & AI for Economists

Lecture 9: Web Scraping and Crawling

Zhaopeng Qu
Business School, Nanjing University

November 13 2025

Roadmap

Today's Agenda

1. Understanding Web Scraping & Crawling
    - What is web scraping vs. web crawling?
    - Why economists need these skills
    - Legal and ethical considerations
2. Web Fundamentals for Scraping
    - How websites work (HTML, CSS, JavaScript)
    - HTTP requests and responses
    - Browser Developer Tools

1. Python Tools for Web Scraping
    - `Requests` library basics
    - `BeautifulSoup` for HTML parsing
    - Setting up your first scraper
2. Practical Example: Static Web Scraping
    - Building a simple scraper
    - Extracting structured data from a static website
    - Handling common challenges in web scraping

Understanding Web Scraping & Crawling

What is Web Scraping/Crawling?

- Web Scraping/Crawling: The automated process of extracting specific data from web pages

- You are building a robot that can crawl the web and extract specific data from it.

Key Characteristics of Web Scraping

- Targeted data extraction: Extract specific data from web pages
- Structured output: Output data in a structured format (CSV, JSON, database)
- Focused on specific information: Focus on specific information rather than the entire page
- Goal-oriented process: The process is goal-oriented, rather than exploratory

Key Characteristics of Web Crawling

- Link discovery: Follow links to other pages
- Automated navigation: Navigate the web and find new pages
- Broad coverage: Crawl a large number of pages

Scraping vs. Crawling

Web Scraping

Focus: Data extraction

Process:

1. Send HTTP request to the target page
2. Parse the HTML content
3. Extract specific elements
4. Structure the data
5. Store results

Packages:

- `BeautifulSoup` for HTML parsing
- `Requests` for HTTP requests
- `pandas` for data storage

Web Crawling

Focus: Page discovery

Process:

1. Send HTTP request to the seed URLs
2. Download page content
3. Extract all links
4. Add new URLs to the queue
5. Repeat the process until the queue is empty

Packages:

- `Scrapy` for web crawling
- `Requests` for HTTP requests
- `pandas` for data storage

Why Economists Need Web Scraping/Crawling

Advantages of Web Scraping/Crawling

Real-time Data

- Monitor prices continuously
- Track policy announcements
- Capture market sentiment

Novel Data Sources

- Social media discussions
- Online reviews
- Job postings
- Housing listings

Granular Information

- Product-level prices
- Firm-level data
- Geographic detail
- High-frequency observations

Cost-Effective

- Free or low-cost access
- Customizable collection
- Reproducible methods

- Especially for China's situation, the **administrative data** is often not available or not accessible to the public.

Cases of Web Scraping/Crawling

Some Cases of Web Scraping/Crawling

Chen, T., & Kung, J. K. (2018). Busting the "Princelings": The Campaign Against Corruption in China's Primary Land Market*. The Quarterly Journal of Economics, 134(1), 185–226.

- The authors scraped the land transaction data from the China Land Market Information System (CLMIS) to study the impact of the "Princelings" on the primary land market.

> *Using data on over a million land transactions during 2004–2016 where local governments are the sole seller, we find that firms linked to members of China's supreme political elites—the Politburo—obtained a price discount ranging from 55.4% to 59.9% compared with those without the same connections...*

Some Cases of Web Scraping/Crawling

Chen, J., Feng, Y., Lu, J., & Qu, Z. (2024). Decoding China's Industrial Policies: A Text Mining Approach*. National Bureau of Economic Research, Working Paper No. 33814.

- The authors scraped the industrial policy documents from official websites at all levels of government, including central, provincial,and municipal administrations.

> *To ensure exhaustive coverage and capture the most recent policy developments, we supplement this data set with continuous web scraping. This process involved systematically collecting data from official websites at all levels of government, including central, provincial,and municipal administrations. The scraping effort targeted websites of key ministries, departments, and other government entities, ensuring a diverse and up-to-date representation of policy documents.*

Legal and Ethical Considerations

Is Web Scraping Legal?

Short Answer: **It depends and with caution.**

Legal Considerations

- `robots.txt`: A website's rules for crawlers
- `Terms of Service`: Website usage agreements
- `Copyright`: Data ownership and fair use
- `Computer Fraud Laws`: Unauthorized access concerns
- `Personal data protection`: Personal data protection laws

Best Practices

✓ Check robots.txt before scraping ✓ Read Terms of Service for restrictions ✓ Respect rate limits - don't overwhelm servers ✓ Use public data only - avoid paywalls and proprietary data ✓ Anonymize personal data if necessary ✓ Cite data sources in your paper/report

The robots.txt File

What is robots.txt?

A file that websites use to communicate crawling permissions to bots

Example robots.txt

```
User-agent: *                # 指定规则适用于所有爬虫（* 表示通配符）
Disallow: /private/          # 禁止爬虫访问 /private/ 目录及其子目录
Disallow: /admin/            # 禁止爬虫访问 /admin/ 目录及其子目录
Crawl-delay: 10              # 设置爬虫请求间隔为 10 秒，避免服务器过载

User-agent: Googlebot        # 以下规则仅适用于 Google 爬虫
Allow: /                     # 允许 Google 爬虫访问所有目录
Sitemap: https://example.com/sitemap.xml  # 指定网站地图位置，帮助爬虫发现页面
```

Web Fundamentals for Scraping

How Websites Work

The Request-Response Cycle

```
You (Browser) → HTTP Request → Web Server
                                   ↓
                            HTML/CSS/JS
                                   ↓
You (Browser) ← HTTP Response ← Web Server
```

What Happens When You Visit a Website?

1. Browser sends HTTP request to server
2. Server processes request and retrieves content
3. Server sends HTTP response with HTML
4. Browser renders HTML to display page
5. Browser requests additional resources (CSS, JS, images)
6. Page fully loads and becomes interactive

HTML Basics

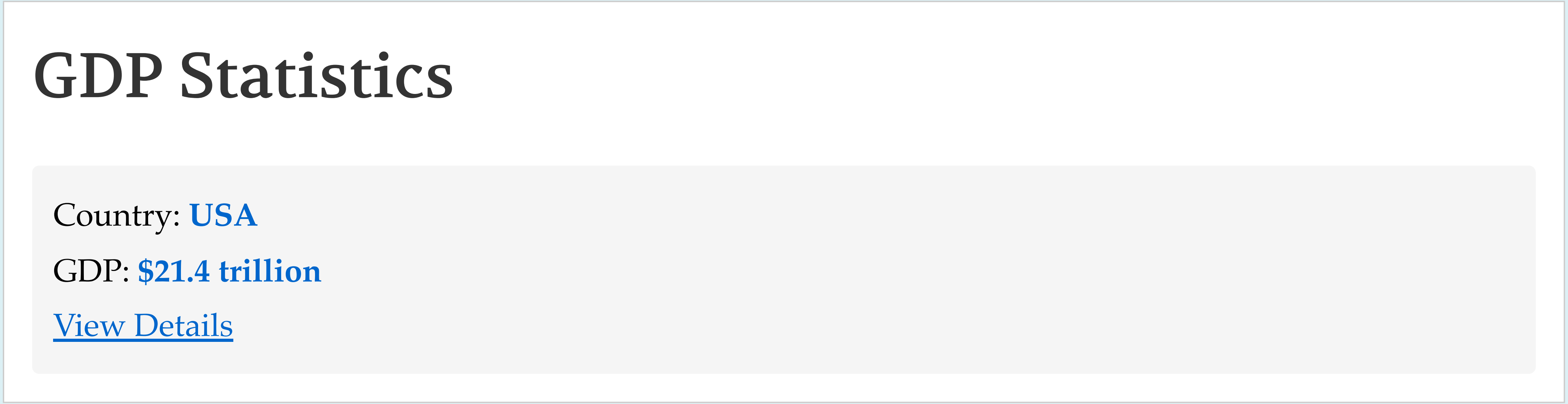
HTML: The Structure of Web Pages

HTML (HyperText Markup Language) uses tags to structure content

Simple HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Economic Data Portal</title>
  </head>
  <body>
    <h1>GDP Statistics</h1>
    <div class="data-table">
      <p>Country: <span class="country">USA</span></p>
      <p>GDP: <span class="gdp">$21.4 trillion</span></p>
      <a href="/details">View Details</a>
    </div>
  </body>
</html>
```

Rendering Effect of the HTML Example


GDP Statistics

Country: **USA**

GDP: **$21.4 trillion**

View Details

Key HTML Elements for Scraping

- `<div>` : Container for content blocks
- `<span>` : Inline container for text
- `<a>` : Links (href attribute contains URL)
- `<table>` : Tables with rows and columns
- `<ul>/<li>` : Lists

HTML Attributes and Classes

```html
<a href="https://example.com" id="link1" class="external">
  Click here
</a>
```

Common Attributes:

- `href`: Link destination
- `id`: Unique identifier
- `class`: Style class (can select multiple elements)
- `src`: Source for images/scripts
- `title`: Additional information

Why Attributes Matter for Scraping

- Classes and IDs help target specific elements
- href contains URLs to follow
- data-* attributes often hold structured data

```python
# Select elements by class
prices = soup.find_all('span', class_='price')

# Select element by ID
title = soup.find('h1', id='product-title')
```

CSS Selectors

CSS: Styling Web Pages

CSS (Cascading Style Sheets) defines how HTML elements look

CSS Selector Syntax

Selector	**Meaning**	**Example**
`tag`	Select by tag name	`div`, `p`, `a`
`.class`	Select by class	`.price`, `.title`
`#id`	Select by ID	`#main-content`
`tag.class`	Tag with class	`span.price`
`tag#id`	Tag with ID	`div#header`
`tag[attr]`	Tag with attribute	`a[href]`

CSS Selector Examples

```css
/* Select all paragraphs with class 'description' */
p.description

/* Select span inside div with class 'product' */
div.product span

/* Select links with href containing 'amazon' */
a[href*="amazon"]
```

Static vs. Dynamic Websites

Static Website

- HTML sent directly from server
- Content visible in "View Source"
- Easy to scrape with Requests

Example:

```
<div class="price">$19.99</div>
```

All content present in initial HTML

Dynamic Website

- JavaScript loads content after page loads
- Content not in initial HTML
- Requires browser automation

Example:

```
<div id="price"></div>
<script>
  // Price loaded via API
  loadPrice('product-123');
</script>
```

Content appears after JavaScript executes

HTTP Requests and Responses

HTTP Request Methods

Method	**Purpose**	**Example Use**
GET	Retrieve data	Load a web page, search results
POST	Submit data	Login forms, file uploads
PUT	Update data	Modify user profile
DELETE	Remove data	Delete a post

For scraping, we mainly use GET requests

HTTP Requests and Responses

HTTP Status Codes

Code	**Meaning**	**Action**
200	Success	Data retrieved successfully
301/302	Redirect	Follow new URL
403	Forbidden	Access denied (check robots.txt)
404	Not Found	Page doesn't exist
429	Too Many Requests	Rate limit exceeded
500	Server Error	Server problem (try again later)

Browser Developer Tools: Chrome

Your Best Friend for Scraping

- It is a built-in tool in the browser that allows you to inspect the HTML, CSS, and JavaScript of a web page.
- The most popular browser is **Chrome**.

How to Open:

- Windows/Linux: `F12` or `Ctrl+Shift+I`
- Mac: `Cmd+Option+I`

Browser Developer Tools: Chrome

Your Best Friend for Scraping

Elements/Inspector

- View HTML structure
- See element classes and IDs
- Test CSS selectors
- Copy XPath

Network

- Monitor HTTP requests
- See API endpoints
- Check response data
- Analyze headers

Console

- Test JavaScript
- Debug errors
- Run quick tests

Application/Storage

- View cookies
- Check local storage
- Examine session data

Lab: Scraping a Static Website