How Non-Coding Journalists Can Build Web Scrapers With AI

Administrator | 11/11/2025 | OSINT

Data has become the backbone of investigative reporting.
Whether tracking government contracts, analyzing environmental violations, or uncovering financial networks, **data-driven journalism** helps reporters verify facts, follow the money, and reveal hidden connections.

Yet one of the biggest challenges isn't interpreting data — it's **getting it**.
When datasets are locked behind complex websites or clunky government portals, reporters often face hours of repetitive copy-pasting just to collect basic information.

That's where **AI-assisted web scraping** changes the game.

From Manual Work to Machine Assistance

Traditionally, building a web scraper required strong programming skills.
Now, with **Large Language Models (LLMs)** such as ChatGPT, Gemini, or Claude, even non-technical journalists can automate data collection using **natural-language prompts** instead of code.

The concept is simple:

> You describe what you need. The AI writes the code.

With a few examples and clear instructions, you can extract tables, text, and documents from online sources, automatically save them as spreadsheets, and repeat the process across hundreds of pages.

Step 1 — Check How the Target Website Works

Before you ask an AI to write a scraper, you need to know **how the data appears** on the page.
Websites come in two main types:

• Static pages: data is directly visible in the page's HTML code.
• Dynamic pages: data is generated by scripts and only appears after interactions such as searches or clicks.

Here's a quick test:

1. Open the page.
2. Right-click → View Page Source.

3. Use Ctrl+F to search for the data you want.If you find it, the page is static. If not, it’s dynamic — and you’ll need an AI to simulate clicks and waiting times.

Step 2 — Ask the AI to Build the Scraper (Static Pages)

Let’s say you want to scrape all press releases from a company’s website.

Your prompt could be:

> “Write a Python script to scrape all press releases (title, date, link) from this page: [URL]. Store the data in a CSV file. The site has pagination.”

The AI (for example, ChatGPT with browsing) will generate:

• a ready-to-run Python script,
• installation steps for required packages (requests, beautifulsoup4),
• and even pagination logic to loop through all pages.

You can copy the code into **Visual Studio Code** or **Google Colab**, run it, and export your data instantly — no programming background required.

Step 3 — Handling Dynamic Websites

Dynamic sites — often used by governments or databases — load data only after you perform certain actions.

Here’s what to tell the AI:

• The exact URL and fields you must interact with (dropdowns, search buttons, etc.).
• The HTML elements (with class or ID attributes) that hold the data.
• The table or list structure of the results.

The AI will likely use **Selenium** or **Playwright** to control a virtual browser.
These libraries simulate human behavior: clicking, typing, and scrolling until the data appears.

> Example: São Paulo’s public procurement database requires users to fill in nine search boxes. An AI can write a Python script that selects the right options, runs the search, waits for results to load, and scrapes each row — repeating the process across 5,000+ pages.

Step 4 — Build Layered Scrapers for Complex Databases

Many online databases use **multi-page hierarchies**:
one page lists projects, another shows project details, and a third displays events or documents.

The best approach is to:

1. Build one scraper to collect the list of links.
2. Feed those URLs to a second scraper to open each sub-page.
3. Optionally, add a third scraper to extract files, attachments, or specific keywords.

This modular setup is easier to debug and lets you expand your investigation step by step.

Step 5 — Troubleshoot and Overcome Blocks

Eventually, your scraper will hit a wall — an error like *“403 Forbidden”* or *“Too Many Requests.”*
Don’t panic. These are common.

Here are three quick fixes AI can help you with:

1. Geolocation blocks

Use a **VPN** (ProtonVPN, NordVPN) or **residential proxies** (Oxylabs) to access region-restricted sites.

2. Request frequency

Add random delays between requests (time.sleep() in Python) or simulate scrolling to look more human.

3. Bot detection

Ask the AI to set realistic **HTTP headers** (User-Agent, language, cookies) so your scraper mimics a real browser.

LLMs can even modify your script automatically when you paste an error message — making them powerful debugging partners.

Step 6 — Schedule and Automate Your Scraper

Once your scraper works, you can make it run **on autopilot**.
Platforms like **GitHub Actions** let you schedule scripts in the cloud — daily, weekly, or hourly — without keeping your computer on.

Prompt example:

> “I have a Python scraper in my GitHub repository.
> Show me how to run it once a day at 1 PM UTC using GitHub Actions.”

The AI will return a ready-to-use YAML configuration file and setup steps.

This approach turns your one-off script into a **continuous monitoring system** — ideal for tracking price changes, tenders, or new government documents.

When Journalism Meets Automation

The Pulitzer Center team has used these techniques to investigate:

- shark meat procurement by Brazil’s government,
- the hidden financial mechanisms behind Amazon deforestation,
- and opaque ride-hailing algorithms in Southeast Asia.

What once required weeks of manual work can now be done in hours — with higher transparency and reproducibility.

As AI continues to evolve, **data journalism is no longer limited to coders**.

The next generation of reporters will combine human judgment with machine precision to uncover patterns no one else can see.

□□ Key Takeaways

Step	What You Do	AI’s Role
1	Identify if the page is static or dynamic	Explains structure
2	Describe what you need	Generates Python scraper
3	Handle interactions (dynamic sites)	Writes Selenium code
4	Combine multiple layers	Manages complex workflows
5	Troubleshoot errors	Suggests fixes automatically
6	Automate with GitHub Actions	Builds cloud scheduler

□□AI is not replacing investigative journalists — it’s **amplifying their reach**.
Knowing how to command an LLM to build a scraper is now as essential as knowing how to use Excel or FOIA requests.

What matters is curiosity, ethics, and persistence — the same qualities that have always defined good reporting.
Data has become the backbone of investigative reporting.
Whether tracking government contracts, analyzing environmental violations, or uncovering financial networks, **data-driven journalism** helps reporters verify facts, follow the money, and reveal hidden connections.

Yet one of the biggest challenges isn’t interpreting data — it’s **getting it**.
When datasets are locked behind complex websites or clunky government portals, reporters often face hours of repetitive copy-pasting just to collect basic information.

That’s where **AI-assisted web scraping** changes the game.

From Manual Work to Machine Assistance

Traditionally, building a web scraper required strong programming skills.
Now, with **Large Language Models (LLMs)** such as ChatGPT, Gemini, or Claude, even non-technical journalists can automate data collection using **natural-language prompts** instead of code.

The concept is simple:

> You describe what you need. The AI writes the code.

With a few examples and clear instructions, you can extract tables, text, and documents from online sources, automatically save them as spreadsheets, and repeat the process across hundreds of pages.

Step 1 — Check How the Target Website Works

Before you ask an AI to write a scraper, you need to know **how the data appears** on the page.
Websites come in two main types:

• Static pages: data is directly visible in the page’s HTML code.
• Dynamic pages: data is generated by scripts and only appears after interactions such as searches or clicks.

Here’s a quick test:

1. Open the page.
2. Right-click → View Page Source.
3. Use Ctrl+F to search for the data you want.If you find it, the page is static. If not, it’s dynamic — and you’ll need an AI to simulate clicks and waiting times.

Step 2 — Ask the AI to Build the Scraper (Static Pages)

Let’s say you want to scrape all press releases from a company’s website.

Your prompt could be:

> “Write a Python script to scrape all press releases (title, date, link) from this page: [URL]. Store the data in a CSV file. The site has pagination.”

The AI (for example, ChatGPT with browsing) will generate:

• a ready-to-run Python script,
• installation steps for required packages (requests, beautifulsoup4),
• and even pagination logic to loop through all pages.

You can copy the code into **Visual Studio Code** or **Google Colab**, run it, and export your data instantly — no programming background required.

Step 3 — Handling Dynamic Websites

Dynamic sites — often used by governments or databases — load data only after you perform certain actions.

Here’s what to tell the AI:

• The exact URL and fields you must interact with (dropdowns, search buttons, etc.).
• The HTML elements (with class or ID attributes) that hold the data.
• The table or list structure of the results.

The AI will likely use **Selenium** or **Playwright** to control a virtual browser.
These libraries simulate human behavior: clicking, typing, and scrolling until the data appears.

> Example: São Paulo’s public procurement database requires users to fill in nine search boxes. An AI can write a Python script that selects the right options, runs the search, waits for results to load, and scrapes each row — repeating the process across 5,000+ pages.

Step 4 — Build Layered Scrapers for Complex Databases

Many online databases use **multi-page hierarchies**:
one page lists projects, another shows project details, and a third displays events or documents.

The best approach is to:

1. Build one scraper to collect the list of links.
2. Feed those URLs to a second scraper to open each sub-page.
3. Optionally, add a third scraper to extract files, attachments, or specific keywords.

This modular setup is easier to debug and lets you expand your investigation step by step.

Step 5 — Troubleshoot and Overcome Blocks

Eventually, your scraper will hit a wall — an error like *“403 Forbidden”* or *“Too Many Requests.”*
Don’t panic. These are common.

Here are three quick fixes AI can help you with:

1. Geolocation blocks

Use a **VPN** (ProtonVPN, NordVPN) or **residential proxies** (Oxylabs) to access region-restricted sites.

2. Request frequency

Add random delays between requests (time.sleep() in Python) or simulate scrolling to look more human.

3. Bot detection

Ask the AI to set realistic **HTTP headers** (User-Agent, language, cookies) so your scraper mimics a real browser.

LLMs can even modify your script automatically when you paste an error message — making them powerful debugging partners.

Step 6 — Schedule and Automate Your Scraper

Once your scraper works, you can make it run **on autopilot**.
Platforms like **GitHub Actions** let you schedule scripts in the cloud — daily, weekly, or hourly — without keeping your computer on.

Prompt example:

> “I have a Python scraper in my GitHub repository.
> Show me how to run it once a day at 1 PM UTC using GitHub Actions.”

The AI will return a ready-to-use YAML configuration file and setup steps.

This approach turns your one-off script into a **continuous monitoring system** — ideal for tracking price changes, tenders, or new government documents.

When Journalism Meets Automation

The Pulitzer Center team has used these techniques to investigate:

• shark meat procurement by Brazil’s government,
• the hidden financial mechanisms behind Amazon deforestation,

• and opaque ride-hailing algorithms in Southeast Asia.

What once required weeks of manual work can now be done in hours — with higher transparency and reproducibility.

As AI continues to evolve, **data journalism is no longer limited to coders**.
The next generation of reporters will combine human judgment with machine precision to uncover patterns no one else can see.

□□ Key Takeaways

Step	What You Do	AI's Role
1	Identify if the page is static or dynamic	Explains structure
2	Describe what you need	Generates Python scraper
3	Handle interactions (dynamic sites)	Writes Selenium code
4	Combine multiple layers	Manages complex workflows
5	Troubleshoot errors	Suggests fixes automatically
6	Automate with GitHub Actions	Builds cloud scheduler

□□AI is not replacing investigative journalists — it's **amplifying their reach**.
Knowing how to command an LLM to build a scraper is now as essential as knowing how to use Excel or FOIA requests.

What matters is curiosity, ethics, and persistence — the same qualities that have always defined good reporting.