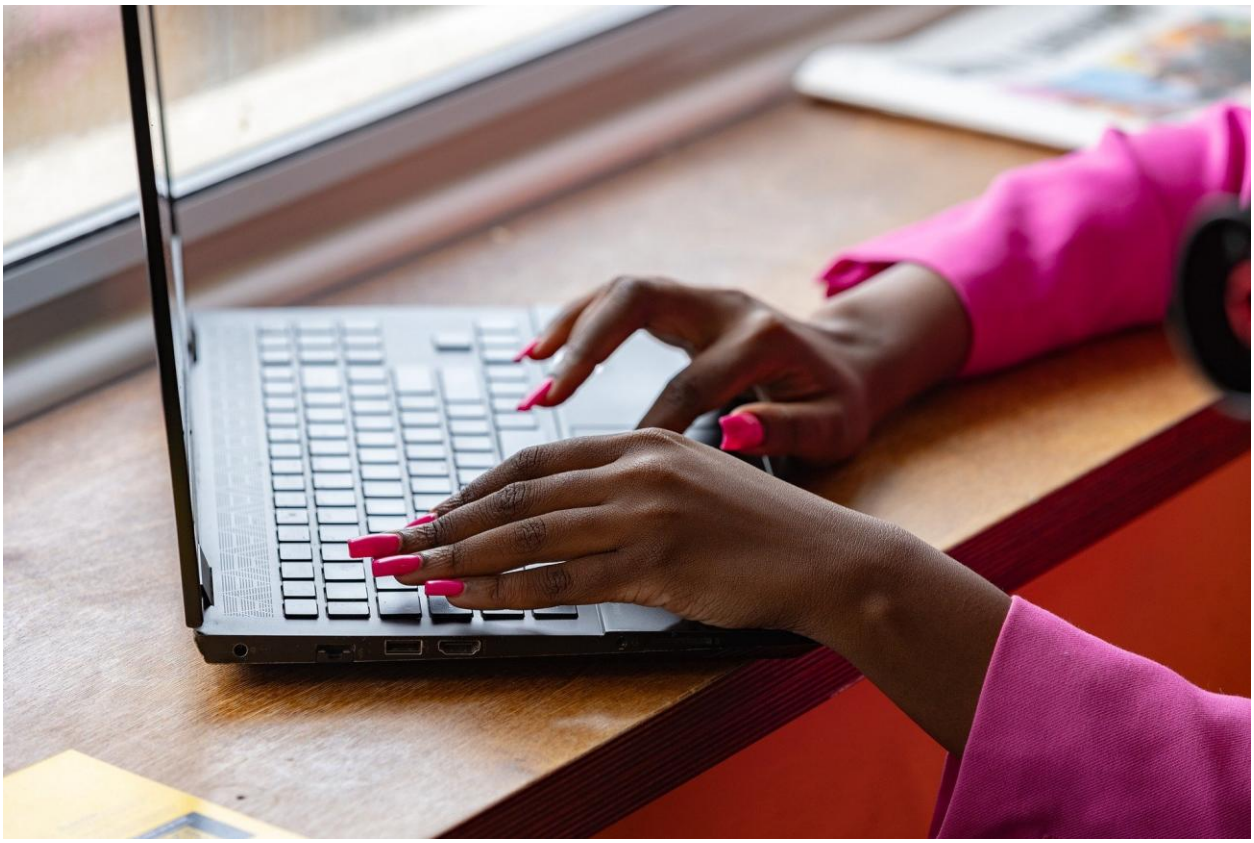
🚀 Day 7 of #30DaysofWebScraping: Data Storage and TrueCar.com Integration

Today, I tackled a real-world web scraping project: scraping "**TrueCar.com**" car listings. This wasn't just about extracting data; I explored how to handle the scraped data efficiently by integrating storage solutions like Excel and PostgreSQL for analysis. Reverse engineering APIs was also part of the project, a concept that never fails to impress me with its ability to simplify complex scraping tasks.



What I Did Today

1. Scraped Car Listings from "TrueCar.com"

TrueCar.com is a very dynamic website and a lot of the data is loaded asynchronously (with JavaScript). Rather than scraping in the traditional way, I reverse-engineered the backend powering the website through APIs. After a bit of poking around the browser's Developer Tools (primarily the Network tab), I found the API endpoint used to request Car listings.

API Data Extraction:

- **Extracted car details like:**
  - Model and make.
  - Price.
  - Mileage.
  - Year
  - Brand
- Used Python's requests library to send API requests and retrieve JSON data.
- Transformed the JSON data into a structured format using pandas.

2. Saved Data to an Excel Sheet

I exported this in an Excel file for easy sharing and quick access. With pandas, I structured the parsed details nicely into a table. This led us to realize that availability of accessible formats for non-technical stakeholders was important.

Why Excel?

- Great for small to medium datasets.
- The data is easily visualizable and filterable for immediate insights.
- Something widely used for sharing data between the teams.

3. Stored Data in PostgreSQL

The highlight of the day was integrating the scraped data into a relational database. I used PostgreSQL, a robust and scalable database management system, to store the car listings.

- **Steps Involved:**
  - Set up a PostgreSQL database locally.
  - Created a table schema to store the car data (e.g., columns for model, price, mileage, etc.).
  - Used psycopg2 to connect Python to the PostgreSQL database and insert the scraped data.
- **Why PostgreSQL?**
  - Offers advanced querying capabilities.
  - Ideal for managing large datasets efficiently.
  - Ensures data consistency and makes future updates seamless.

4. Ran SQL Queries to Analyze the Data

Once the data was stored in PostgreSQL, I ran SQL queries to gain insights from the dataset. Some examples of the analyses I performed:

- Counted the number of listings by car model.
- Identified the most expensive and least expensive cars in the dataset.
- Filtered cars based on mileage to find budget-friendly options.

This step emphasized the power of combining web scraping with SQL for comprehensive data analysis.

Challenges Faced

1. **API Authentication:**
    - The API required specific headers like User-Agent and tokens for access. Debugging these requirements through the browser's Developer Tools was critical.
2. **Database Schema Design:**
    - Structuring the data schema in PostgreSQL to handle different car attributes efficiently took some planning.
3. **Data Validation:**
    - Ensuring that no duplicate records were stored in the database required adding unique constraints.

Reflections & What's Next

Day7 was a strong remind of the fact that web scraping is more than just download data, it is about making this data usable and accessible. Using PostgreSQL or similar solutions allows for more complex queries and data persistence.

Up next, the anti-bot measures and CAPTCHA challenges— because every step forward unlocks another layer of the web's complexity. Let's keep exploring, one API call at a time! 💻✨