Report on Automated Scraping of Daily Prices Data from FCA Website

Akash Kundu

August 24, 2025

Introduction

This project focused on automating the process of collecting **Daily Prices** data from the **FCA InfoWeb portal** (`https://fcainfoweb.nic.in/reports/report_menu_web.aspx`). The website provides tabular daily commodity price reports, but the data can only be retrieved one date at a time after solving a captcha. Since the dataset of interest spanned multiple months (February–April 2020), we designed a robust scraping workflow using Python, Selenium, and Excel integration.

Workflow Overview

The workflow consisted of several distinct stages:

Step 1: **Navigating to the Website** The scraper first launches the FCA InfoWeb portal using Selenium and the Chrome WebDriver.

Step 2: **Selecting Report Type** On the portal, the following selections are made:

- Choose *Price Report*.
- From the dropdown, select *Daily Prices*.

Step 3: **Inputting Date and Solving Captcha** For each target date, the script:

- Clears the date input field and enters the required date.
- Waits for manual user input to solve the captcha and press "Get Data".

Step 4: **Extracting Tabular Data** Once the table loads, the script scrapes all rows and columns displayed for that date. The date is also stored alongside each row to maintain temporal alignment.

Step 5: **Incremental Saving** To avoid data loss in case of crashes or interruptions:

- Each scraped batch (corresponding to one date) is appended directly to an Excel file.
- This ensures the file always contains progress up to the most recent successful scrape.

Step 6: Handling Missing Dates Some dates occasionally failed to load due to site or captcha issues. For these cases:

- A separate script was created to target only the missing dates.
- Before scraping, the script checks which dates already exist in the Excel file to avoid duplicates.

Step 7: Post-Processing the Dataset After the scraping stage, additional cleaning was applied:

- The first row of data (which usually contained the table headers) was extracted and used as the global column names for the dataset.
- The *Date* column was converted into proper date format.
- All rows were sorted chronologically to create a consistent time series dataset.
- The final cleaned dataset was exported into a new Excel file.

Python Script Responsibilities

To keep the project modular, different Python files were designed for specific tasks:

- `scraping.py` Handles the main scraping logic:
  - Launches Selenium and navigates to the FCA InfoWeb portal.
  - Inputs dates, waits for captcha solving, and scrapes tabular data.
  - Saves scraped results incrementally into an Excel file.
- `missing_dates.py` Focuses on handling missing or failed dates:
  - Accepts a list of user-provided dates.
  - Checks which dates are already present in the Excel file.
  - Scrapes only the missing dates and appends them to the dataset.
- `fca_clean_sort.py` Performs post-processing of the collected dataset:
  - Reads the combined Excel file.
  - Uses the first row of data as column headers.
  - Ensures the *Date* column is in correct date format.
  - Sorts the entire dataset chronologically.
  - Outputs a clean, analysis-ready Excel file.

Results

By following the above workflow:

- Daily commodity price data was collected for the period of February–April 2020.
- Missing dates were successfully handled and appended without duplication.
- The final dataset was organized in Excel, with correct headers and chronological ordering.

Reproducibility: Steps to Recreate the Project

If a professor or colleague wishes to reproduce this work, the following steps should be followed:

Step 1: Set up the Environment

- Install Python (3.9+ recommended).
- Install Google Chrome.
- Create a virtual environment: `python -m venv .venv`
- Activate it: `.venv\Scripts\activate` (Windows).
- Install dependencies:

```
pip install selenium pandas openpyxl webdriver-manager
```

Step 2: Run Initial Scraper Execute `scraper.py` to scrape data for a given list of dates. Example: scrape February to April 2020 in bulk.

Step 3: Handle Missing Dates If any dates failed to scrape, run `retry_missing.py` and pass only those dates. The script will append data without creating duplicates.

Step 4: Clean and Sort Dataset Run `clean_and_sort.py` to:

- Apply the proper headers (from the first scraped table).
- Ensure the *Date* column is in correct format.
- Sort all rows chronologically.

Step 5: Final Output The cleaned dataset is saved into a new Excel file (e.g., `daily_prices_cleaned.xlsx`) ready for analysis.

Conclusion

The project successfully automated what would otherwise be a highly repetitive manual task. Key achievements included:

- Automation of navigation and data extraction from a captcha-protected portal.

- Incremental saving for reliability.
- Post-processing for consistent, analysis-ready data.
- Modular Python scripts for scraping, retrying missing dates, and cleaning.

This workflow can be extended to scrape additional time periods or adapted for similar government data portals.