Web Scraping Project Report

1. Overall Approach and Design

For this project, I selected a publicly accessible website featuring a structured listing format to demonstrate practical web scraping. The chosen site includes a tabular or card-style presentation of items (e.g., vehicle for selling listings), which made it suitable for structured data extraction.

The scraping was implemented using **Python** with the **Scrapy framework/BeautifulSoup**, which is robust, scalable, and allows modular development. The scraper was designed to:

- Crawl the main listing pages
- Follow pagination using a for loop with control of min max page numbers to extract data by replacing the url query strings
- Extract targeted fields from each record
- Export the data in both **JSON** formats

Key data fields included in the output were:

- Title
- Price
- Location
- Days
- Mileage

The Scrapy spider was structured to follow a modular pipeline:

- **Parsing**: Extract data from the listing elements
- **Output**: Save to a JSON/CSV using Scrapy's feed export feature

2. Challenges Handling

a. Dynamic Content (JavaScript)

The target site is not loaded with additional records dynamically using JavaScript (AJAX-based infinite scroll). To address such a situation, we can use Selenium, a headless browser automation tool that simulates user interactions.

- Selenium with a **headless Chrome** driver was integrated to load dynamic content.
- Once the page was fully loaded, the relevant HTML was parsed using **BeautifulSoup** to extract the structured data.

Alternatively, for performance optimization on large-scale scraping, **Scrapy-Splash** or **Playwright** can be considered for full JS rendering.

b. Rate Limiting / Anti-Scraping

We can follow responsible scraping practices as below to avoid rate limits and/ Anti-scraping:

- Introduced **randomized delays** between requests (`time.sleep()` with random intervals)
- Used **custom User-Agent headers** to avoid detection
- Enabled **AutoThrottle** in Scrapy settings to adapt crawl speed
- Considered **rotating proxies** for scale-out scenarios, though not applied here due to the small dataset size

c. Followed following setting and options in this project to make sure to avoid Rate Limiting / Anti-Scraping

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False

RETRY_HTTP_CODES = [429, 503]
RETRY_TIMES = 5


DOWNLOAD_DELAY = 3  # 3 seconds between requests
AUTOTHROTTLE_ENABLED = True
```

```
AUTOTHROTTLE_START_DELAY = 3
AUTOTHROTTLE_MAX_DELAY = 10
AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0

CONCURRENT_REQUESTS = 1
```

---

3. Analysis & Summary

a. Data Summary

From the scraping run, a total of **476 records** were collected. The dataset contained structured information across consistent fields.

b. Insights

- **Duplicates**: A few duplicate entries appeared due to repeat listings—this was resolved by filtering based on unique URLs
- **Completeness**: Most fields were consistently populated; however, some optional metadata fields were occasionally missing
- **Consistency**: HTML structure was consistent across pages, simplifying the extraction logic

The final dataset can be readily used for analysis or ingestion into dashboards or ML workflows.

---

4. Deliverables

- **Codebase**: Available on GitHub - my-scrap-project / https://github.com/kavindaktk2025/my-scrap-project.git
- **Data Output**: JSON files is included in the repository(under folder output) - https://github.com/kavindaktk2025/my-scrap-project/blob/main/output/output.json
- **Instructions**: A `README.md` with step-by-step instructions on running the scraper is provided