Web Scrapping Report

Tools and Technologies

- **Python**: Programming language used for scripting.
- **Selenium**: Web automation tool used to handle dynamic JavaScript content.
- **BeautifulSoup**: Library used for parsing HTML content.
- **Edge WebDriver**: Browser driver for Edge used with Selenium.
- **Pandas**: Library for data manipulation and analysis.
- **Jupyter notebook** : for code execution

Scrapped Website: Cars24.com

Objective: To scrape detailed car listings from cars24.com, capturing various attributes including car name, year of registration, kilometers driven, fuel type, transmission type, discount price, actual price, discount offered, model, and test drive location.

Scrapped Detailes:

1. Car brand (Maruti)
2. Model(Alto ....)
3. Registered_year (2016)
4. km_driven(97,698KM)
5. Fuel_type(petrol)
6. Transmission_type(Manual)
7. Add_ons(low run cost)
8. Discount_price(2.31 lakh)
9. Actual_price(2.60 lakh)
10. Discount_offered(28.76k off)
11. Test_drive_location(chhatarpur, Delhi)

Process

1. Initial Scraping Attempt

- **Tool Used**: BeautifulSoup
- **Issue**: Only 20 entries were retrieved as the site loads additional content dynamically using JavaScript.
- **Diagnosis**: Disabling JavaScript revealed that only a limited set of data is visible without JavaScript, confirming dynamic content loading.

2. Switch to Selenium

- **Reason**: To handle dynamic content loading and JavaScript-driven updates.
- **Approach**: Implemented scrolling and refreshing to load all entries.

Challenges and Solutions

1. **Dynamic Content Loading**:
    - **Challenge**: The website uses JavaScript to dynamically load additional data, which is not visible in the initial HTML source.
    - **Solution**: Used Selenium to automate scrolling and reloading of the page to ensure all content is loaded and accessible.
2. **Handling Multiple Data Points**:
    - **Challenge**: Different pieces of information were located in various parts of the HTML, requiring multiple passes and different parsing strategies.
    - **Solution**: Implemented a series of data extraction steps, each focusing on different attributes of the car listings.
3. **Synchronization Issues**:
    - **Challenge**: Ensuring that data extraction steps are synchronized with page loading and content visibility.
    - **Solution**: Added delays and multiple page refreshes to make sure all content was loaded before extracting data.

Conclusion

The code successfully automates the process of scraping comprehensive car listing data from cars24.com by addressing the challenges associated with dynamic content loading. The final dataset includes detailed attributes of each car.