**Project Report: Mobile Scraping using Python**

**1. Introduction**

The project involves scraping product information from an e-commerce website that sells mobile phones. The goal is to collect data such as the product's title, rating, number of reviews, discounted price, original price, and discount percentage from multiple pages on the website. The scraped data is then stored in a CSV file for further analysis.

This project demonstrates the use of **web scraping** techniques to extract data from an HTML page and process it using Python libraries such as requests, BeautifulSoup, and pandas.

**2. Tools and Libraries Used**

* **requests**: A library used to send HTTP requests and fetch data from the website.
* **BeautifulSoup (bs4)**: A library for parsing HTML and XML documents. It helps extract the required information from the webpage.
* **re**: Python’s regular expressions library for pattern matching (though it's not heavily used in this specific implementation).
* **pandas**: A data manipulation library for storing and processing structured data. It is used to organize the scraped data and save it into a CSV file.
* **time**: A Python library used to introduce delays between scraping different pages to avoid overloading the server (via time.sleep).

**3. Class Structure and Key Functions**

The MobileScraper class is designed to handle the scraping process. Here is an overview of its structure:

**3.1 Initialization (\_\_init\_\_)**

The class constructor initializes the following:

* base\_url: The base URL of the website from which to scrape data.
* max\_pages: The maximum number of pages to scrape.
* Several lists to store the data: title\_list, rating\_list, reviews\_list, discount\_price\_list, original\_price\_list, and off\_on\_every\_mobile\_list.
* HTTP headers (header) to simulate a browser request to avoid being blocked by the website.

**3.2 Scrape Method (scrape)**

The scrape method is responsible for:

* Iterating through multiple pages of the website (from page 1 to max\_pages).
* Sending HTTP requests to fetch the page content.
* Using BeautifulSoup to parse the page and extract the product data.
* Extracting product information by calling the extract\_product\_data method for each product.

**3.3 Extract Product Data (extract\_product\_data)**

The extract\_product\_data method:

* Extracts various pieces of data for each product:
  + **Title**: Extracts the product title.
  + **Rating**: Extracts the product rating.
  + **Reviews**: Extracts the number of reviews.
  + **Discount Price**: Extracts the discounted price of the product.
  + **Original Price**: Extracts the original price of the product.
  + **Discount Percentage**: Extracts the discount percentage for the product.
* The extracted data is appended to the corresponding list (e.g., title\_list, rating\_list, etc.).
* Conditional logic is applied to handle products where discount price or original price might be missing or invalid.

**3.4 Save Data to CSV (save\_to\_csv)**

The save\_to\_csv method:

* Creates a pandas.DataFrame with the scraped data.
* Saves the DataFrame to a CSV file (default: mobile\_data.csv).
* Prints a message to confirm the data has been saved successfully.

**4. How the Code Works**

1. **Initialization**: The MobileScraper object is created with the base URL (https://priceoye.pk/mobiles) and the number of pages to scrape (max\_pages = 23).
2. **Scraping**:
   * The scrape method is called, which iterates through each page (page=1 to page=23) of the website.
   * For each page, a GET request is made to the URL, and the HTML content is fetched.
   * The BeautifulSoup object is created to parse the HTML response and find all product details.
   * For each product, the extract\_product\_data method is invoked to extract the product title, rating, reviews, discount price, original price, and discount percentage.
3. **Data Processing**:
   * For each product, the scraped data is stored in respective lists (e.g., title\_list, rating\_list).
   * The discount\_price and original\_price are processed to remove commas and convert them into integers. If the discount\_price is less than the original\_price, both are stored; otherwise, the discount price is considered the original price.
   * The discount percentage is extracted and stored in off\_on\_every\_mobile\_list.
4. **Saving Data**: After scraping all pages, the save\_to\_csv method is called to save the data in CSV format.
5. **User-Agent Header**: The code includes a custom user-agent string in the headers to mimic a real browser, making the requests less likely to be blocked by the website.
6. **Delays Between Requests**: A 3-second delay (time.sleep(3)) is added between page requests to prevent overloading the server and being blocked by the website.

**5. Challenges and Considerations**

* **Rate Limiting**: The website might block requests if it detects scraping behavior, especially when making frequent requests in a short period. To mitigate this, a delay (time.sleep(3)) is added between each request.
* **Missing or Incorrect Data**: Sometimes, the discount\_price, original\_price, or other product attributes might be missing. The code accounts for such cases by assigning default values (e.g., 0 if a price is not found).
* **HTML Structure**: The scraping logic is heavily dependent on the structure of the HTML content. If the website's layout changes, the scraping logic might need to be updated.
* **Data Storage**: Storing the data in a CSV format makes it easier to analyze and share the results. However, for larger datasets, a database might be more suitable.

**6. Results**

The script successfully scrapes product data from multiple pages of the website and stores it in a CSV file. The scraped data includes:

* Product titles
* Ratings
* Reviews
* Discount prices
* Original prices
* Discount percentages

This data can be used for further analysis, such as identifying the most popular products, analyzing pricing trends, or understanding the discounts offered by different brands.

**7. Future Improvements**

* **Error Handling**: Implement better error handling to catch network issues, invalid responses, or missing data.
* **Data Cleaning**: Improve data cleaning by ensuring all extracted prices are formatted correctly and handle edge cases where prices may not be present.
* **Pagination Handling**: Automate detection of the total number of pages (instead of hardcoding max\_pages), in case the number of pages changes in the future.
* **Concurrency**: Introduce parallel processing (e.g., using concurrent.futures or multiprocessing) to scrape pages faster.

**8. Conclusion**

This project successfully demonstrates how to scrape data from an e-commerce website using Python. It utilizes libraries like requests, BeautifulSoup, and pandas to fetch, parse, and store product information. The data collected can be useful for various analyses, and the code can be extended to handle larger datasets or more complex scraping tasks.