**Web Scraping via Selenium**

Objective:

1. Scrape Comments of any product, and apply Sentiment analysis.
2. Scrape at least 80 products, their “name”, “price”, “Starts”, “Reviews”.
3. Scrape the following requirements for any type of product:
   1. Top 10 Sellers store names
   2. Daraz Mall Store/ Private store
   3. Total "Mobile Covers" listings on his store
   4. Avg Prices
   5. Avg Units selling
   6. Positive Seller Ratings
   7. Seller Country

**Scrape Comments of any product, and apply Sentiment analysis (part 1)**

In the first part of our project, we employed Selenium to scrape product data from Daraz, a popular online marketplace. The goal was to collect comprehensive information about products, including names, prices, and ratings, for use in subsequent data analysis. To achieve this, we set up a Python environment with Selenium, Pandas, and Webdriver Manager, ensuring that our web scraping framework was robust and compatible with the latest browser versions.

Using the Chrome WebDriver, we navigated to the Daraz website and implemented a scraping script to locate and extract product details. We targeted specific HTML elements corresponding to product names, prices, and ratings. The script handled dynamic content by allowing sufficient time for the page elements to load before extraction.

Furthermore, we implemented pagination handling to scrape multiple pages of products. By clicking the 'Next' button and repeating the extraction process for each new page, we ensured a thorough collection of data across numerous product listings. This iterative scraping continued until all accessible pages were processed.

The collected data was organized into a list of dictionaries, each representing a product with its name, price, and rating. This list was then converted into a Pandas DataFrame for easy manipulation and analysis. Finally, we saved the DataFrame to a CSV file, creating a structured dataset ready for further exploration and analysis. The WebDriver session was closed to conclude the scraping process.

**Analysising the Data (part b)**

This code we've got here is all about processing some user reviews and figuring out their sentiment. First off, we're using NLTK, which is handy for natural language processing stuff. We grab some stopwords and punctuation from English, because we don't need those messing up our analysis. Then, we dive into the reviews stored in a DataFrame called Review\_df. We clean them up by getting rid of stopwords, punctuation, and any weird characters that aren't letters or numbers. Everything gets converted to lowercase for consistency. After all that cleaning, we slap the results into a new DataFrame named Processed\_df. This new DataFrame keeps track of the original reviewer's name, the cleaned-up review, and then we bring in NLTK's Vader sentiment intensity analyzer to calculate sentiment scores for each review. With a quick lambda function, we tack on those sentiment scores to each review in the DataFrame. Boom, there you have it – a nifty pipeline for preprocessing and analyzing text data to dig into user sentiment.

**Scrape at least 80 products (part 2)**

In the second part of our project, we used Selenium to scrape detailed product information from the Daraz website. Our goal was to collect comprehensive data, including product names, prices, and ratings, to create a dataset for further analysis.

We began by setting up the necessary environment, installing Selenium for web scraping, Pandas for data manipulation, and Webdriver Manager for managing the Chrome WebDriver. This setup ensured compatibility with the latest browser versions, simplifying the scraping process.

Using the Chrome WebDriver, we navigated to the Daraz homepage and allowed sufficient time for the page to load. The core task involved locating product elements and extracting relevant information. We targeted specific HTML elements to capture data on product names, prices, and ratings. Missing ratings were marked as 'No rating' to maintain consistency.

Handling pagination was crucial for scraping data from multiple pages. The script found and clicked the 'Next' button to navigate through successive pages, pausing briefly to ensure each page fully loaded before continuing extraction. This iterative approach enabled the collection of a large volume of product information.

The extracted data was organized into a list of dictionaries, each representing a product with its details. This list was converted into a Pandas DataFrame for structured storage and analysis. Finally, the DataFrame was saved to a CSV file, creating a comprehensive dataset for future use.

**Part 3)**

The Selenium script automates the collection of seller information from an e-commerce platform, likely Daraz, based on user input. The user inputs a product type, and the script initializes lists to store data on seller names, average prices, seller ratings, country of origin, follower counts, and whether the seller is part of Daraz Mall or a private entity.

The WebDriver, using Chrome, opens the specified URL, locates the search bar, inputs the query, and retrieves search results. The script loops through multiple pages and products, collecting details such as seller name, price, shipping country, follower count, seller type, and rating, avoiding duplicates.

The script handles pop-ups, navigates between pages, and manages errors. Collected data is stored in a pandas DataFrame (Q3\_df) for further analysis. The DataFrame includes columns for seller names, Daraz Mall or private status, average prices, average unit selling prices, seller ratings, country of origin, and follower counts.