Project Name: Jumia Product Data Scraper

1. Project Overview

Description

This project is a **web scraping pipeline** designed to extract product details and customer reviews from **Jumia Egypt**. The scraper collects metadata such as product names, SKUs, URLs, prices, and customer reviews. The primary focus is on **data collection only** - the project does not include analysis or insights generation. The final output is saved in a structured **CSV file** for potential future analysis.

2. Project Objectives

The primary objectives of this project are:

1. **Data Collection**:
    - Scrape product details (name, SKU, URL, price, etc.) and customer reviews from **Jumia Egypt**.
    - Handle pagination and content extraction to ensure comprehensive data collection.
2. **Data Structuring**:
    - Store the scraped data in a structured format (CSV) for potential future use.
3. **Technical Implementation**:
    - Develop a reliable scraping pipeline using Python libraries.
    - Implement proper error handling and request management.

3. Scope of the Project

Target Website

- **Website**: Jumia Egypt
- **Categories Scraped**: General product search results (not category-specific)
- **Data Collected**:
    - **Product Metadata**: Name, SKU, URL, Overall Rating
    - **Customer Reviews**: Review text, Rating, Date, Author

Limitations

- The scraper focuses on collecting raw data only, with no analysis component
- The project collects data from the first 10 pages of search results
- No translation or multilingual processing is implemented in this version

4. Tools and Technologies Used

Programming Languages and Libraries

- **Python**: Primary programming language

- **Requests**: For sending HTTP requests to the website
- **BeautifulSoup**: For parsing HTML content
- **Pandas**: For handling and saving data in CSV format
- **time**: For implementing request delays

Ethical and Legal Considerations

- Respected **robots.txt** guidelines of the website
- Added delays (`time.sleep`) between requests to avoid overloading the server
- Ensured proper User-Agent headers to identify the scraper
- Collected only publicly available information

5. Data Collection Process

Step 1: Website Structure Analysis

- Used browser developer tools to inspect Jumia's HTML structure
- Identified key elements for product links (`<a class="core">`), SKUs (`<ul class="-pvs -mvxs -phm -lsn">`), and review sections

Step 2: Scraper Implementation

- Developed modular functions for different scraping tasks:
  - `get_product_links`: Extracts product URLs from search results pages
  - `extract_sku`: Retrieves the SKU from individual product pages
  - `extract_reviews_and_rating`: Collects customer reviews and ratings
  - `scrape_product`: Coordinates the scraping of a single product page
  - `save_csv`: Formats and saves the collected data to CSV

Step 3: Handling Technical Challenges

- **Pagination**: Implemented page iteration to collect data from multiple result pages
- **Duplicate Prevention**: Used Python sets to avoid duplicate product URLs
- **Request Management**: Added 1-second delays between requests to prevent blocking
- **Error Handling**: Implemented checks for missing elements (like SKUs)

6. Technical Implementation Details

Core Functions

Product Link Collection

```python
def get_product_links(category_url, pages=1):
    """
    Get product links from Jumia category pages.
    """
    links = set()  # Use a set to avoid duplicates

    for p in range(1, pages + 1):
        url = f"{category_url}&page={p}"
        resp = requests.get(url, headers)
        soup = BeautifulSoup(resp.text, "html.parser")  # Parse the HTML

        for a in soup.select("a.core"):  # Product cards
            href = a.get("href")
            if href:
                links.add("https://www.jumia.com.eg" + href.split("?")[0])  # Full
URL without query params

        time.sleep(1)

    return list(links)
```

Data Extraction Process

1. First, the scraper collects product URLs from search results pages
2. For each product URL, it extracts:
    - Product name from `<h1 class="-fs20">`
    - SKU by parsing the product details list
    - Customer reviews by constructing a special reviews URL using the SKU
3. Review data is extracted from the reviews page using CSS selectors

Data Storage

```
def save_csv(data, fname="jumia_reviews.csv"):
    """
    Save scraped data to CSV.
    """
    rows = []
    for p in data:
        revs = p["reviews"] or [{}]
        for r in revs:
            rows.append({
                "product_name": p["name"],
                "url": p["url"],
                "sku": p["sku"],
                "overall_rating": p["rating"],
                "rating": r.get("rating"),
                "headline": r.get("headline"),
                "review": r.get("review"),
                "date": r.get("date"),
                "author": r.get("author")
            })

    df = pd.DataFrame(rows).dropna()
    df.to_csv(fname, index=False, encoding="utf-8-sig")
    print(f"[INFO] Saved {len(rows)} rows to {fname}")
    return df
```

7. Output Format

The scraper produces a **CSV file** with the following fields:

Field Name	Description	Example Value
product_name	Name of the product	"iPhone 14 Pro Max"
url	Product page URL	"https://www.jumia.com.eg/iphone..."
sku	Product SKU code	"12345"
overall_rating	Average product rating	"4.8"
rating	Individual review rating	"5"
headline	Review headline/title	"Great Phone"
review	Full review text	"Excellent performance!"
date	Date of the review	"2023-10-01"
author	Author of the review	"Ahmed M."

8. Challenges Faced

1. **Website Structure Complexity**:
   Jumia's HTML structure is complex with multiple nested elements, requiring careful selector identification.

2. **Missing Data Elements**:
   Some products don't have SKUs listed, requiring error handling to skip problematic products.

3. **Request Rate Limiting**:
   Without proper delays, the website would block the scraper after multiple requests.

4. **URL Construction**:
   The reviews page requires a specific URL format using the product SKU.

Solutions Implemented

- Used comprehensive CSS selectors to navigate the HTML structure
- Added error checking for missing elements (like SKUs)
- Implemented 1-second delays between requests
- Created robust URL construction logic for reviews pages

9. Future Work

1. **Category-Specific Scraping**:
   Extend the scraper to target specific product categories (Electronics, Fashion, etc.)

2. **Enhanced Error Handling**:
   Improve robustness with retry mechanisms for failed requests

3. **Data Validation**:
   Add checks to verify data integrity before saving
4. **Scalability Improvements**:
   Implement parallel processing to increase scraping speed
5. **Storage Options**:
   Add support for database storage (SQL or NoSQL) in addition to CSV

10. Conclusion

This project successfully implemented a web scraping pipeline for collecting product data from Jumia Egypt. The scraper reliably extracts product metadata and customer reviews, handling pagination and potential errors in the process. The output is a clean, structured CSV file containing raw data ready for potential future analysis. The project demonstrates proper web scraping techniques while respecting ethical considerations and website limitations.

The focus remains strictly on data collection - no analysis or insights generation is performed by this pipeline, as it is designed to provide raw data for potential future use cases.
