SECP3133: HIGH PERFORMANCE DATA PROCESSING

SECTION 02

Project Proposal

Optimizing High-Performance Data Processing for Large-Scale Web Crawlers

Faculty of Computing

No	Nama	No Matrik
1	Soh Han Wei	A22EC0273
2	Loo Jia Chang	A22EC0074
3	Muhammad Nur Azhar Bin Mohd Yazid	A22EC0220
4	Nur Arini Fatihah Binti Mohd Sabir	A22EC0244

Table of Content

1.0 Introduction

1.1 Project Background

Today, data is growing and expanding at an alarming rate, especially on the Internet. Millions of websites currently provide valuable information across different fields. E-commerce websites such as Lazada are one of the most data-rich platforms, providing a variety of product lists, prices, reviews and other important details. Crawling or extracting these data on a large scale is very important for various commercial purposes, including price comparison, market analysis and consumer sentiment analysis. However, large-scale web crawling brings great challenges, such as managing server load, ensuring the efficiency of crawling and abiding by moral principles.

In this project, we aim to optimize the web crawling process for large-scale data extraction by using high performance computing (HPC) technology. The main focus in this project is to efficiently crawl websites and extract structured data while overcoming performance bottlenecks and moral problems.

1.2 Objective

- To design and implement an optimized high-performance web crawler that can extract large-scale structured data from Lazada efficiently without overloading the server.
- To analyze and evaluate the performance of different data extraction frameworks (Scrapy, BeautifulSoup, Selenium, Request-HTML) in terms of speed, scalability, and resource consumption for real-time and bulk crawling tasks.

1.3 Target Web - Lazada

For our project, we've chosen Lazada Malaysia as the primary website for data extraction. Lazada dominates Malaysian markets via its extensive catalog of available listings covering various categories such as electronics, fashion, home appliances, and beauty products. The platform stands out to users because its streamlined product pages enhance bulk data extraction processes.

One of the main reasons we picked Lazada is due to the features it offers that are relevant to our project. Each product listing on Lazada usually includes detailed information such as product id, product name, current and original pricing, discount percentages, seller names and ratings, stock status, shipping options, and average rating. These fields are exactly the kind of data we aim to collect, as they are useful not just for testing our crawler's performance but also for practical data analysis tasks later on.

The product data through Lazada becomes more reliable because the platform features both official stores and verified sellers. User-generated reviews at Lazada operate at a high pace because a large number of customers provide ratings and feedback that prove useful for future analysis projects.

The platform lets users control their product crawl by searching and filtering to maintain focus on targeted data while avoiding server congestion. Our implementation of crawling ethics becomes possible through these system controls that allow us to follow both page crawl delays and reduce wasteful page requests.

Overall, Lazada provides a rich and structured data environment that aligns well with our technical goals.The platform offers genuine web data processing difficulties and consistent structured information that enables performance assessment of our high-speed crawler system.

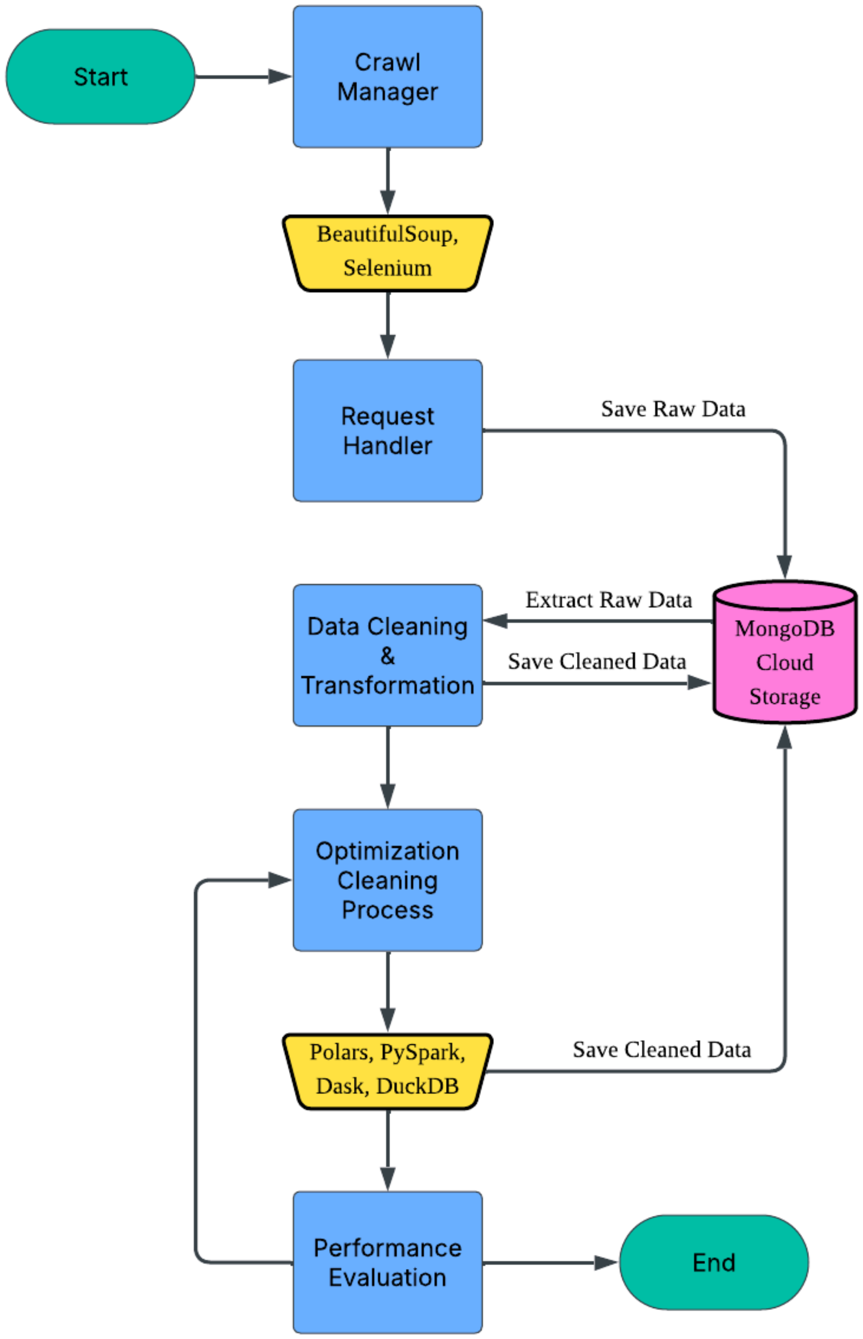
1.4 Data To Be Extracted

To make sure we gather over 100,000 structured data points without overwhelming the site with too much crawling, we’ll zero in on the most valuable fields from product listings, reviews, and seller profiles. We’ve crafted a streamlined extraction plan aimed at collecting data on 10,000 to 20,000 products, focusing on 10 to 15 key fields for each. We'll also include additional seller and review information to help us scale effectively.

Fields to Extract (Per Product)		
Field	**Example**	**Data Type**
product_name	"Xiaomi Redmi Note 12"	String
price	"RM 799.00"	String/Float
location	"China"	String
numbersold	"1.7k sold"	String
rating	"(20)"	String

2.0 System Design & Architecture

2.1 Crawler's Architecture



Description of Components

1. **Crawl Manager**
    a. Orchestrates the crawling strategy, manages URL queues, monitors request frequency, and invokes crawling engines
    b. Ensures respect for rate limits and robots.txt

2. **Crawling Libraries - BeautifulSoup, Selenium**
    a. **Selenium**: used for rendering dynamic JavaScript-based content
    b. **BeautifulSoup**: parses the static HTML/XML content to extract relevant fields

3. **Request Handler**
    a. Handles complex headers and user-agent rotation for Lazada.
    b. Manages cookies and session tokens (important for logged-in-only data)
    c. Captures anti-bot flags and implements fallback (retry or proxy)

4. **Data Cleaning & Transformation Plan**
    a. Prepare raw data for analysis through basic preprocessing and feature formatting using Pandas
    b. Key Operation:
        i. **Duplicate Removal**: Eliminates redundant rows
        ii. **Missing Value Handling**:
            1. Replace missing strings with "unknown"
            2. Fills critical numerical fields with "0"
        iii. **Data Formatting**
            1. Convert "5K sold" to "5000" via regex
            2. Extracts numbers from "(100)" format in Number of Ratings
            3. Strips non-numeric characters from Price and handles "unknown" edge cases
        iv. **Type Conversion**
            1. Price -> float (when valid), otherwise "unknown"
            2. Quantity Sold, Number of Rating -> integer

5. **Data Storage -MongoDB**
    a. Persistently **stores raw and cleaned datasets**
    b. Uses MongoDB Atlas with pymango for CRUD operations
    c. Supports insertion of Pandas DataFrame dictionaries and flexible schema

6. **Performance Evaluation**

a. Quantifies efficiency improvements from the optimized data cleaning processes
b. Metric captured (**time taken for key operations**, **CPU and memory footprint**)
c. Data visualization via **matplotlib** and **seaborn**

2.2 Libraries and Framework

Softwares / Tools	**Usage / Functionality**
Google Docs	Documentation & Report
Lucidchart	Architecture Design
VS Code, Google Colab	Coding Environment
Python	Data Crawling, Data Cleaning and Transformation, Data Visualization
Selenium, BeautifulSoup	Web Scraping
MongoDB	Cloud DBMS
.csv File	Data Storage
MongoDB	DBMS Storage
Polars, PySpark, Dask, DuckDB	Optimization Libraries

2.4 Role of each members

Member	Responsibility
Soh Han Wei (Group Leader)	● Coordinate weekly meetings & track milestones in Github ● Seek lecturer approval and handle external communications ● Maintain master project timeline & risk register ● Perform final review of all GitHub pull‑requests and deliverables
Loo Jia Chang	● Design crawler architecture ● Implement core crawling logic with robots.txt & rate‑limiting ● Set up progressive data checkpoints (CSV / SQLite) ● Collaborate on introducing threading / asyncio for speed‑up
Muhammad Nur Azhar Bin Mohd Yazid	● Draft planning brief and documentation templates ● Write unit & integration tests for crawler and processing modules ● Validate dataset integrity after each run (duplicates, missing values)

	● Compile Turnitin‑ready report and design presentation slides
Nur Arini Fatihah Binti Mohd Sabir	● Clean & transform raw data, ensuring ≥ 100, 000 records ● Apply at least two optimization techniques (asyncio, threading, Spark/Dask) ● Benchmark CPU, memory, and runtime pre‑ vs post‑optimization ● Provide performance metrics and graphs for the final report

3.0 Data Collection

3.1 Crawling Method

1. **Pagination Handling**

   In this scraper, the script uses Selenium to load a starting URL. It determines the number of pages by parsing the pagination elements using BeautifulSoup:

   ```
   pagination = soup.select(".ant-pagination-item")
   total_pages = int(pagination[-1].text) if pagination else 1
   ```

   A 'for' loop serves to process all the pages one by one:

   ```
   for page in range(total_pages):
   ```

   It simulates user behavior by clicking the ‘Next Page’ button using:

   ```
   next_button = driver.find_element(By.CSS_SELECTOR, ".ant-pagination-next > button")
   time.sleep(random.uniform(3, 5))  # Simulate reading delay
   next_button.click()
   ```

   This method allows traversal through all products within paginated lists without encountering any missed items.

2. **Rate-Limiting and Anti-Bot Measures**

   To avoid being blocked by Lazada’s anti-bot systems, the scraper implements basic rate-limiting and random delays. Random delays between 3 to 5 seconds are introduced using time.sleep(random.uniform(...)).

   ```
   time.sleep(random.uniform(3, 5))  # Simulate reading delay
   ```

   The script includes a CAPTCHA detection mechanism:

   ```
   try:
       WebDriverWait(driver, 20).until(
           EC.presence_of_element_located((By.ID, "captcha"))  # Adjust if Lazada uses different selector
       )
       input("⚠️ CAPTCHA detected. Please solve it manually in the browser, then press Enter to continue...")
   except:
       pass
   ```

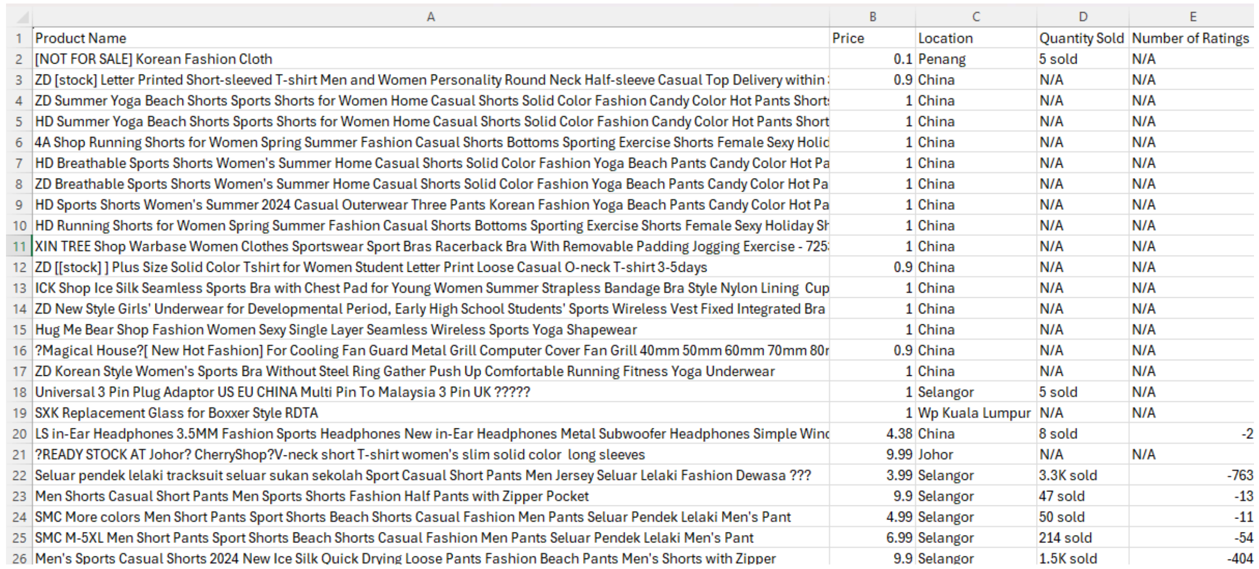
   This lowers the chance of IP blocking by rate limits, though it's still manual in handling CAPTCHAs, which can interrupt continuous scraping.

3. **Asynchronous Support (Not used)**

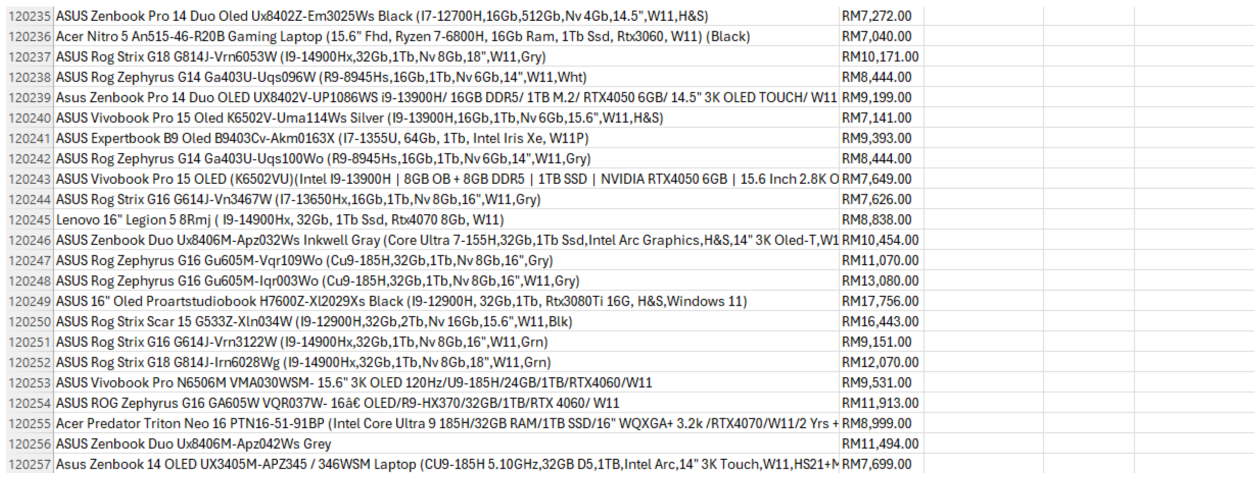
   Currently, the scraper runs synchronously, it processes one page at a time, waiting for each page to load and extract before moving to the next. The implementation process for this method is straightforward however, execution runs slowly because there's no task parallelization. Since Lazada is JavaScript-heavy, asynchronous crawling would need headless browser clusters (e.g., Playwright or Puppeteer in async mode) or headless Selenium grid with concurrency, which adds overhead.

3.2 Number of Records Collected

A total of 120256 rows of data has been extracted.



	A	B	C	D	E
1	Product Name	Price	Location	Quantity Sold	Number of Ratings
2	[NOT FOR SALE] Korean Fashion Cloth	0.1	Penang	5 sold	N/A
3	ZD [stock] Letter Printed Short-sleeved T-shirt Men and Women Personality Round Neck Half-sleeve Casual Top Delivery within	0.9	China	N/A	N/A
4	ZD Summer Yoga Beach Shorts Sports Shorts for Women Home Casual Shorts Solid Color Fashion Candy Color Hot Pants Short	1	China	N/A	N/A
5	HD Summer Yoga Beach Shorts Sports Shorts for Women Home Casual Shorts Solid Color Fashion Candy Color Hot Pants Short	1	China	N/A	N/A
6	4A Shop Running Shorts for Women Spring Summer Fashion Casual Shorts Bottoms Sporting Exercise Shorts Female Sexy Holic	1	China	N/A	N/A
7	HD Breathable Sports Shorts Women's Summer Home Casual Shorts Solid Color Fashion Yoga Beach Pants Candy Color Hot Pa	1	China	N/A	N/A
8	ZD Breathable Sports Shorts Women's Summer Home Casual Shorts Solid Color Fashion Yoga Beach Pants Candy Color Hot Pa	1	China	N/A	N/A
9	HD Sports Shorts Women's Summer 2024 Casual Outerwear Three Pants Korean Fashion Yoga Beach Pants Candy Color Hot Pa	1	China	N/A	N/A
10	HD Running Shorts for Women Spring Summer Fashion Casual Shorts Bottoms Sporting Exercise Shorts Female Sexy Holiday Sh	1	China	N/A	N/A
11	XIN TREE Shop Warbase Women Clothes Sportswear Sport Bras Racerback Bra With Removable Padding Jogging Exercise - 725	1	China	N/A	N/A
12	ZD [[stock]] Plus Size Solid Color Tshirt for Women Student Letter Print Loose Casual O-neck T-shirt 3-5days	0.9	China	N/A	N/A
13	ICK Shop Ice Silk Seamless Sports Bra with Chest Pad for Young Women Summer Strapless Bandage Bra Style Nylon Lining Cup	1	China	N/A	N/A
14	ZD New Style Girls' Underwear for Developmental Period, Early High School Students' Sports Wireless Vest Fixed Integrated Bra	1	China	N/A	N/A
15	Hug Me Bear Shop Fashion Women Sexy Single Layer Seamless Wireless Sports Yoga Shapewear	1	China	N/A	N/A
16	?Magical House?[New Hot Fashion] For Cooling Fan Guard Metal Grill Computer Cover Fan Grill 40mm 50mm 60mm 70mm 80r	0.9	China	N/A	N/A
17	ZD Korean Style Women's Sports Bra Without Steel Ring Gather Push Up Comfortable Running Fitness Yoga Underwear	1	China	N/A	N/A
18	Universal 3 Pin Plug Adaptor US EU CHINA Multi Pin To Malaysia 3 Pin UK ?????	1	Selangor	5 sold	N/A
19	SXK Replacement Glass for Boxxer Style RDTA	1	Wp Kuala Lumpur	N/A	N/A
20	LS in-Ear Headphones 3.5MM Fashion Sports Headphones New in-Ear Headphones Metal Subwoofer Headphones Simple Winc	4.38	China	8 sold	-2
21	?READY STOCK AT Johor? CherryShop?V-neck short T-shirt women's slim solid color long sleeves	9.99	Johor	N/A	N/A
22	Seluar pendek lelaki tracksuit seluar sukan sekolah Sport Casual Short Pants Men Jersey Seluar Lelaki Fashion Dewasa ???	3.99	Selangor	3.3K sold	-763
23	Men Shorts Casual Short Pants Men Sports Shorts Fashion Half Pants with Zipper Pocket	9.9	Selangor	47 sold	-13
24	SMC More colors Men Short Pants Sport Shorts Beach Shorts Casual Fashion Men Pants Seluar Pendek Lelaki Men's Pant	4.99	Selangor	50 sold	-11
25	SMC M-5XL Men Short Pants Sport Shorts Beach Shorts Casual Fashion Men Pants Seluar Pendek Lelaki Men's Pant	6.99	Selangor	214 sold	-54
26	Men's Sports Casual Shorts 2024 New Ice Silk Quick Drying Loose Pants Fashion Beach Pants Men's Shorts with Zipper	9.9	Selangor	1.5K sold	-404



120235	ASUS Zenbook Pro 14 Duo Oled Ux8402Z-Em3025Ws Black (I7-12700H,16Gb,512Gb,Nv 4Gb,14.5",W11,H&S)	RM7,272.00			
120236	Acer Nitro 5 An515-46-R20B Gaming Laptop (15.6" Fhd, Ryzen 7-6800H, 16Gb Ram, 1Tb Ssd, Rtx3060, W11) (Black)	RM7,040.00			
120237	ASUS Rog Strix G18 G814J-Vrn6053W (I9-14900Hx,32Gb,1Tb,Nv 8Gb,18",W11,Gry)	RM10,171.00			
120238	ASUS Rog Zephyrus G14 Ga403U-Uqs096W (R9-8945Hs,16Gb,1Tb,Nv 6Gb,14",W11,Wht)	RM8,444.00			
120239	Asus Zenbook Pro 14 Duo OLED UX8402V-UP1086WS i9-13900H/ 16GB DDR5/ 1TB M.2/ RTX4050 6GB/ 14.5" 3K OLED TOUCH/ W11	RM9,199.00			
120240	ASUS Vivobook Pro 15 Oled K6502V-Uma114Ws Silver (I9-13900H,16Gb,1Tb,Nv 6Gb,15.6",W11,H&S)	RM7,141.00			
120241	ASUS Expertbook B9 Oled B9403Cv-Akm0163X (I7-1355U, 64Gb, 1Tb, Intel Iris Xe, W11P)	RM9,393.00			
120242	ASUS Rog Zephyrus G14 Ga403U-Uqs100Wo (R9-8945Hs,16Gb,1Tb,Nv 6Gb,14",W11,Gry)	RM8,444.00			
120243	ASUS Vivobook Pro 15 OLED (K6502VU)(Intel I9-13900H | 8GB OB + 8GB DDR5 | 1TB SSD | NVIDIA RTX4050 6GB | 15.6 Inch 2.8K O	RM7,649.00			
120244	ASUS Rog Strix G16 G614J-Vn3467W (I7-13650Hx,16Gb,1Tb,Nv 8Gb,16",W11,Gry)	RM7,626.00			
120245	Lenovo 16" Legion 5 8Rmj (I9-14900Hx, 32Gb, 1Tb Ssd, Rtx4070 8Gb, W11)	RM8,838.00			
120246	ASUS Zenbook Duo Ux8406M-Apz032Ws Inkwell Gray (Core Ultra 7-155H,32Gb,1Tb Ssd,Intel Arc Graphics,H&S,14" 3K Oled-T,W1	RM10,454.00			
120247	ASUS Rog Zephyrus G16 Gu605M-Vqr109Wo (Cu9-185H,32Gb,1Tb,Nv 8Gb,16",Gry)	RM11,070.00			
120248	ASUS Rog Zephyrus G16 Gu605M-Iqr003Wo (Cu9-185H,32Gb,1Tb,Nv 8Gb,16",W11,Gry)	RM13,080.00			
120249	ASUS 16" Oled Proartstudiobook H7600Z-Xl2029Xs Black (I9-12900H, 32Gb,1Tb, Rtx3080Ti 16G, H&S,Windows 11)	RM17,756.00			
120250	ASUS Rog Strix Scar 15 G533Z-Xln034W (I9-12900H,32Gb,2Tb,Nv 16Gb,15.6",W11,Blk)	RM16,443.00			
120251	ASUS Rog Strix G16 G614J-Vrn3122W (I9-14900Hx,32Gb,1Tb,Nv 8Gb,16",W11,Grn)	RM9,151.00			
120252	ASUS Rog Strix G18 G814J-Irn6028Wg (I9-14900Hx,32Gb,1Tb,Nv 8Gb,18",W11,Grn)	RM12,070.00			
120253	ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K OLED 120Hz/U9-185H/24GB/1TB/RTX4060/W11	RM9,531.00			
120254	ASUS ROG Zephyrus G16 GA605W VQR037W- 16â€ OLED/R9-HX370/32GB/1TB/RTX 4060/ W11	RM11,913.00			
120255	Acer Predator Triton Neo 16 PTN16-51-91BP (Intel Core Ultra 9 185H/32GB RAM/1TB SSD/16" WQXGA+ 3.2k /RTX4070/W11/2 Yrs +	RM8,999.00			
120256	ASUS Zenbook Duo Ux8406M-Apz042Ws Grey	RM11,494.00			
120257	Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM Laptop (CU9-185H 5.10GHz,32GB D5,1TB,Intel Arc,14" 3K Touch,W11,HS21+M	RM7,699.00			

Data field that are recorded from the dataset are :

1. **Product Name** - The title or name of the product listed on Lazada.
2. **Price** - The listed selling price of the product in Malaysian Ringgit (RM).
3. **Location** - The geographical location from which the product is shipped.
4. **Quantity Sold** - The number of units sold for the product, as shown on the listing.

5. **Number of Ratings** - The total number of customer reviews or ratings received by the product.

3.3 Ethical Considerations

In conducting this web crawling activity on Lazada Malaysia, several ethical considerations were taken into account to ensure responsible and respectful data collection. Firstly, the scraping script was designed to simulate human browsing behavior by introducing random delays (time.sleep(random.uniform(2.5, 4.5))) between requests. The strategy controls excessive traffic by limiting request rates, which protects Lazada's servers from reaching capacity levels that could disrupt the website operations.

Secondly, the collected data only consists of already viewable public information, including product names, prices, locations, sales numbers, and customer ratings. The process did not involve any access or storage of personal data along with sensitive or copyrighted information.

Thirdly, the scraper includes a CAPTCHA detection mechanism. When a CAPTCHA appears, the automated script stops and demands user intervention to complete the CAPTCHA. This behavior respects the platform's attempt to control automated access and serves as a safeguard against bypassing security measures.

Finally, the data extracted will be used only for academic and analysis purposes, such as understanding product trends and pricing behaviors. The data will not be used for resale, marketing, or any form of commercial exploitation. Credit is duly acknowledged to Lazada as the data source, and the scraping activities were conducted in alignment with general web scraping best practices and ethical research standards.

4.0 Data Processing

4.1 Cleaning Method

```
# 1. Create connection with MongoDB
```

```
from pymongo import MongoClient

uri =
"mongodb+srv://hanwei:hanwei123@mongodbms-p1.5d52qxu.mongodb.net/?retryW
rites=true&w=majority&appName=MongoDBMS-P1"
client = MongoClient(uri)

db = client["MongoDBMS-P1"]
collection = db["mycollection"]
```

```
# 2. import raw dataset into MongoDB
```

```
import pandas as pd
import json

# Load CSV or JSON
df = pd.read_csv("Dataset.csv", encoding="ISO-8859-1")

# Convert to dictionary format for MongoDB
data_dict = df.to_dict("records")

# Insert into MongoDB
collection.insert_many(data_dict)
```

```
# Verify data has been uploaded successfully
```

```
# View a few documents
for doc in collection.find().limit(5):
    print(doc)
```

```
{'_id': ObjectId('6818c00bd0aa127d11c44e08'), 'Product Name': '[NOT FOR SALE] Korean Fashion Clot
{'_id': ObjectId('6818c00bd0aa127d11c44e09'), 'Product Name': 'ZD [stock] Letter Printed Short-sl
{'_id': ObjectId('6818c00bd0aa127d11c44e0a'), 'Product Name': 'ZD Summer Yoga Beach Shorts Sports
{'_id': ObjectId('6818c00bd0aa127d11c44e0b'), 'Product Name': 'HD Summer Yoga Beach Shorts Sports
{'_id': ObjectId('6818c00bd0aa127d11c44e0c'), 'Product Name': '4A Shop Running Shorts for Women S
```

```
# 3. Load data from MongoDB into DataFrame and drop autogenerated _id
```

```
df = pd.DataFrame(list(collection.find()))

if '_id' in df.columns:
    df.drop(columns=['_id'], inplace=True)
```

```
# 4. Drop duplicates
```

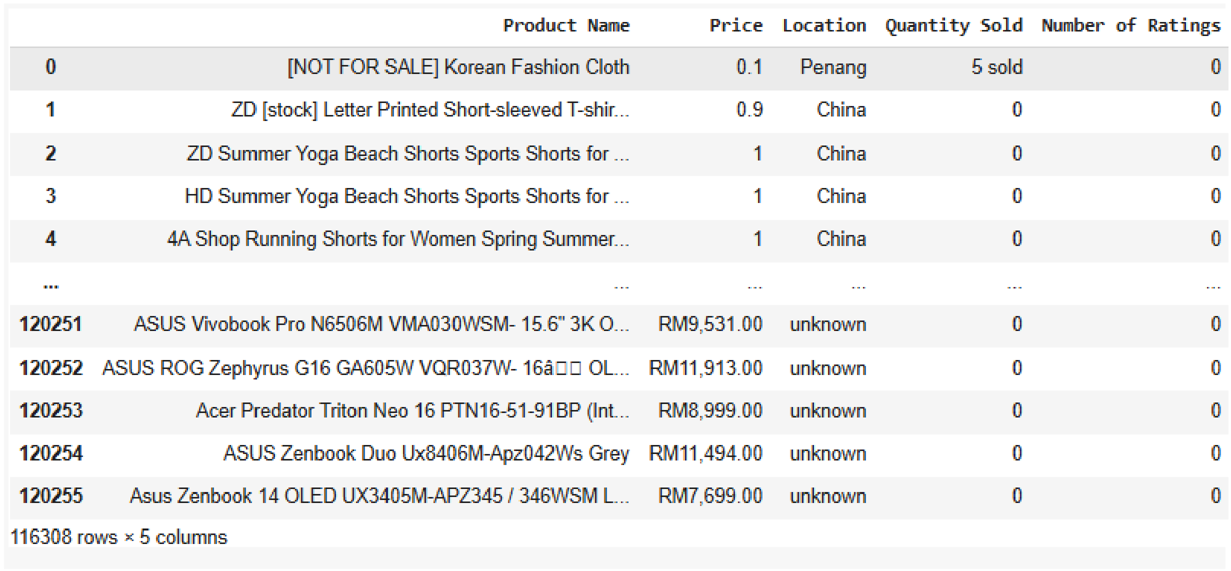
```
df.drop_duplicates(inplace=True)
```

```
                                          Product Name        Price  Location  Quantity Sold  Number of Ratings
0                       [NOT FOR SALE] Korean Fashion Cloth          0.1    Penang         5 sold                NaN
1              ZD [stock] Letter Printed Short-sleeved T-shir...          0.9     China            NaN                NaN
2                  ZD Summer Yoga Beach Shorts Sports Shorts for ...            1     China            NaN                NaN
3                  HD Summer Yoga Beach Shorts Sports Shorts for ...            1     China            NaN                NaN
4                4A Shop Running Shorts for Women Spring Summer...            1     China            NaN                NaN
...                                                            ...          ...       ...            ...                ...
120251   ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...  RM9,531.00       NaN            NaN                NaN
120252  ASUS ROG Zephyrus G16 GA605W VQR037W- 16â OL...  RM11,913.00       NaN            NaN                NaN
120253        Acer Predator Triton Neo 16 PTN16-51-91BP (Int...  RM8,999.00       NaN            NaN                NaN
120254             ASUS Zenbook Duo Ux8406M-Apz042Ws Grey  RM11,494.00       NaN            NaN                NaN
120255    Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...  RM7,699.00       NaN            NaN                NaN
116308 rows × 5 columns
```

```
# 5. Replace NaN in specific columns with "unknown" and fill actual NaN
with "0" in critical columns
```

```
df_copy["Product Name"].fillna("unknown", inplace=True)
df_copy["Location"].fillna("unknown", inplace=True)
df_copy["Price"].fillna("unknown", inplace=True)

df_copy["Quantity Sold"].fillna("0", inplace=True)
```

```
df_copy["Number of Ratings"].fillna("0", inplace=True)
```



```
                                          Product Name        Price  Location  Quantity Sold  Number of Ratings
0                    [NOT FOR SALE] Korean Fashion Cloth          0.1    Penang         5 sold                  0
1             ZD [stock] Letter Printed Short-sleeved T-shir...   0.9     China              0                  0
2              ZD Summer Yoga Beach Shorts Sports Shorts for ...    1     China              0                  0
3              HD Summer Yoga Beach Shorts Sports Shorts for ...    1     China              0                  0
4           4A Shop Running Shorts for Women Spring Summer...       1     China              0                  0
...                                                    ...          ...       ...            ...                ...
120251   ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...  RM9,531.00   unknown              0                  0
120252   ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL...  RM11,913.00   unknown              0                  0
120253         Acer Predator Triton Neo 16 PTN16-51-91BP (Int...  RM8,999.00   unknown              0                  0
120254           ASUS Zenbook Duo Ux8406M-Apz042Ws Grey  RM11,494.00   unknown              0                  0
120255   Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...  RM7,699.00   unknown              0                  0
116308 rows × 5 columns
```

4.2 Data Structure

Column	Non-Null	Count	Dtype
Product Name	116308	non-null	object
Price	116308	non-null	float64
Location	116308	non-null	object
Quantity Sold	116308	non-null	int64
Number of Ratings	116308	non-null	int64

4.3 Transformation and Formatting

```
# 6. Make a copy on dataframe for recover purpose
```

```
df_copy = df.copy()
```

```
# 7. Clean "Quantity Sold" (e.g., "5K sold" → 5000)
```

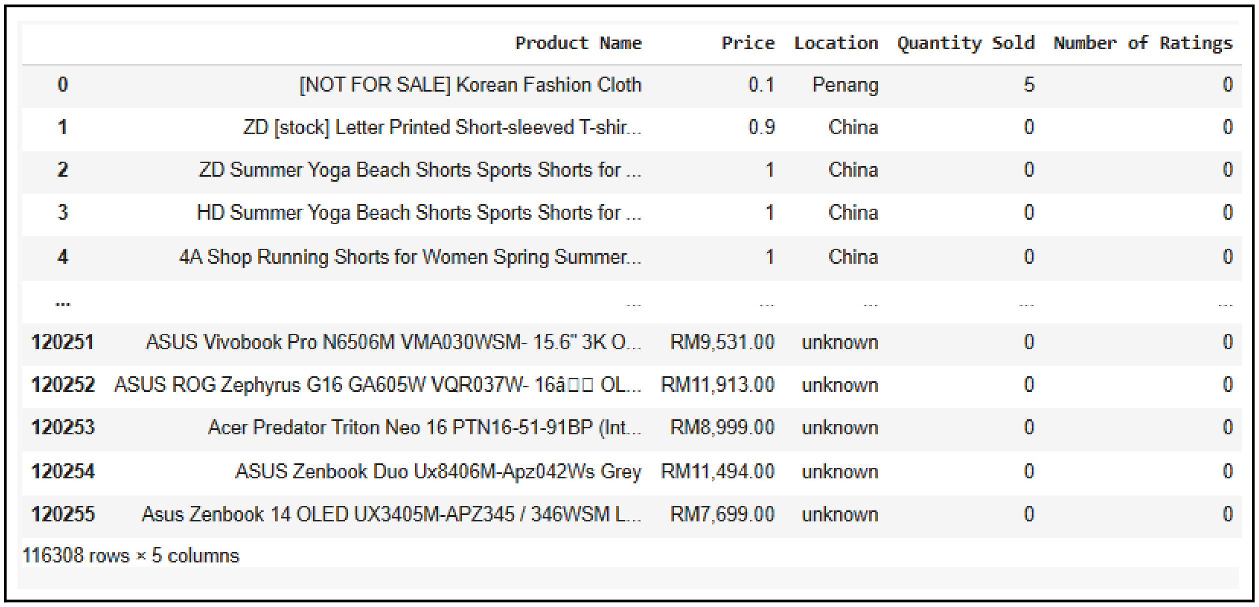
```
import re
```

```
def clean_quantity(q):
    if isinstance(q, str):
        q = q.lower().replace("sold", "").strip()
        if "k" in q:
            return int(float(q.replace("k", "")) * 1000)
        return int(re.findall(r"\d+", q)[0]) if re.findall(r"\d+", q)
else 0
    return 0

df_copy["Quantity Sold"] = df_copy["Quantity
Sold"].apply(clean_quantity)
```

```
                                         Product Name        Price  Location  Quantity Sold  Number of Ratings
0                   [NOT FOR SALE] Korean Fashion Cloth        0.1    Penang              5                  0
1                ZD [stock] Letter Printed Short-sleeved T-shir...   0.9     China              0                  0
2              ZD Summer Yoga Beach Shorts Sports Shorts for ...     1     China              0                  0
3              HD Summer Yoga Beach Shorts Sports Shorts for ...     1     China              0                  0
4             4A Shop Running Shorts for Women Spring Summer...      1     China              0                  0
...                                                        ...     ...       ...            ...                ...
120251   ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...   RM9,531.00   unknown        0                  0
120252  ASUS ROG Zephyrus G16 GA605W VQR037W- 16â OL...  RM11,913.00   unknown        0                  0
120253     Acer Predator Triton Neo 16 PTN16-51-91BP (Int...   RM8,999.00   unknown        0                  0
120254        ASUS Zenbook Duo Ux8406M-Apz042Ws Grey  RM11,494.00   unknown        0                  0
120255  Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...   RM7,699.00   unknown        0                  0
116308 rows × 5 columns
```

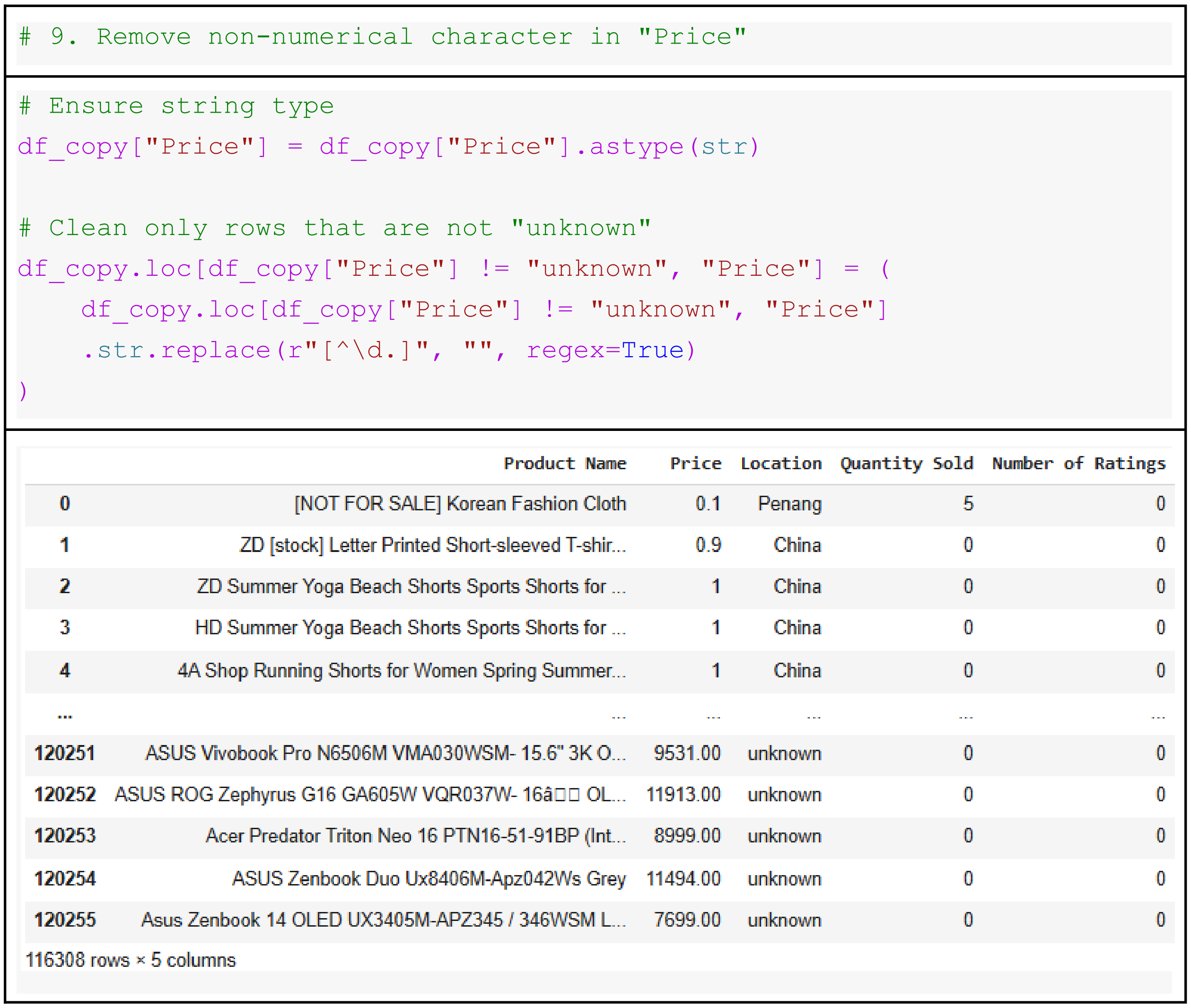
```
# 8. Clean "Number of Ratings" (e.g., "(10)" → 10)
```

```
def clean_ratings(r):
    if isinstance(r, str):
        match = re.search(r"\d+", r)
        return int(match.group()) if match else 0
    return 0

df_copy["Number of Ratings"] = df_copy["Number of
Ratings"].apply(clean_ratings)
```



	Product Name	Price	Location	Quantity Sold	Number of Ratings
0	[NOT FOR SALE] Korean Fashion Cloth	0.1	Penang	5	0
1	ZD [stock] Letter Printed Short-sleeved T-shir...	0.9	China	0	0
2	ZD Summer Yoga Beach Shorts Sports Shorts for ...	1	China	0	0
3	HD Summer Yoga Beach Shorts Sports Shorts for ...	1	China	0	0
4	4A Shop Running Shorts for Women Spring Summer...	1	China	0	0
...	...	...	...	...	...
120251	ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...	RM9,531.00	unknown	0	0
120252	ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL...	RM11,913.00	unknown	0	0
120253	Acer Predator Triton Neo 16 PTN16-51-91BP (Int...	RM8,999.00	unknown	0	0
120254	ASUS Zenbook Duo Ux8406M-Apz042Ws Grey	RM11,494.00	unknown	0	0
120255	Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...	RM7,699.00	unknown	0	0

116308 rows × 5 columns



```
# 9. Remove non-numerical character in "Price"
```

```
# Ensure string type
df_copy["Price"] = df_copy["Price"].astype(str)

# Clean only rows that are not "unknown"
df_copy.loc[df_copy["Price"] != "unknown", "Price"] = (
    df_copy.loc[df_copy["Price"] != "unknown", "Price"]
    .str.replace(r"[^\d.]", "", regex=True)
)
```

	Product Name	Price	Location	Quantity Sold	Number of Ratings
0	[NOT FOR SALE] Korean Fashion Cloth	0.1	Penang	5	0
1	ZD [stock] Letter Printed Short-sleeved T-shir...	0.9	China	0	0
2	ZD Summer Yoga Beach Shorts Sports Shorts for ...	1	China	0	0
3	HD Summer Yoga Beach Shorts Sports Shorts for ...	1	China	0	0
4	4A Shop Running Shorts for Women Spring Summer...	1	China	0	0
...	...	...	...	...	...
120251	ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...	9531.00	unknown	0	0
120252	ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL...	11913.00	unknown	0	0
120253	Acer Predator Triton Neo 16 PTN16-51-91BP (Int...	8999.00	unknown	0	0
120254	ASUS Zenbook Duo Ux8406M-Apz042Ws Grey	11494.00	unknown	0	0
120255	Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...	7699.00	unknown	0	0

116308 rows × 5 columns

```
# 11. Convert "Price" to float and leave the word "unknown", “Quantity
Sold” & “Number of Ratings” to int
```

```
def to_float_or_unknown(val):
    try:
        return float(val)
    except:
        return "unknown"

df_copy["Price"] = df_copy["Price"].apply(to_float_or_unknown)

df_copy["Quantity Sold"] = df_copy["Quantity Sold"].astype(int)
df_copy["Number of Ratings"] = df_copy["Number of Ratings"].astype(int)
```

	Product Name	Price	Location	Quantity Sold	Number of Ratings
0	[NOT FOR SALE] Korean Fashion Cloth	0.1	Penang	5	0
1	ZD [stock] Letter Printed Short-sleeved T-shir...	0.9	China	0	0
2	ZD Summer Yoga Beach Shorts Sports Shorts for ...	1.0	China	0	0
3	HD Summer Yoga Beach Shorts Sports Shorts for ...	1.0	China	0	0
4	4A Shop Running Shorts for Women Spring Summer...	1.0	China	0	0
...	...	...	...	...	...
120251	ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O...	9531.0	unknown	0	0
120252	ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL...	11913.0	unknown	0	0
120253	Acer Predator Triton Neo 16 PTN16-51-91BP (Int...	8999.0	unknown	0	0
120254	ASUS Zenbook Duo Ux8406M-Apz042Ws Grey	11494.0	unknown	0	0
120255	Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L...	7699.0	unknown	0	0

116308 rows × 5 columns

```
# 11. Upload cleaned data back to MongoDB
```

```
collection.drop()
collection.insert_many(df_copy.to_dict("records"))
```

5.0 Optimization Techniques

5.1 Method Used

Libraries	Function
Polars	Optimized data processing through **lazy execution**, **efficient columnar operations**, and **parallelism**, which allow handling and cleaning large datasets in a much more efficient manner
PySpark	Optimizes data processing by **distributing tasks across a cluster of machines**, **enabling parallel execution** and **in-memory computation**. It handles large datasets efficiently with its distributed DataFrame API, which supports operations like filtering, aggregating, and transforming data. PySpark provides scalability and fault tolerance for big data processing.
Dask	Optimizes data processing by **parallelizing operations across multiple CPU cores or machines** using **task scheduling**. It extends familiar libraries like Pandas and NumPy to handle larger-than-memory datasets efficiently through chunked computation and lazy evaluation. Dask enables scalable and interactive data processing workflows for big data on a single machine or distributed cluster.
DuckDB	Optimizes analytical data processing through an in-process columnar engine that supports **vectorized execution** and **multithreading** for parallel query performance. It achieves high efficiency on a single machine by leveraging **parallel execution**, **zero-copy data access**, and **smart caching**.

5.2 Code Overview

5.2.1 Polars

```
# 1. Load data using pandas with proper encoding, then convert to Polars
```

```
try:
    pandas_df = pd.read_csv("Dataset.csv", encoding="ISO-8859-1")
    df = pl.from_pandas(pandas_df)
    df_lazy = df.lazy()

    print("\nSchema of the DataFrame:")
    schema = df_lazy.collect_schema()
    for name, dtype in schema.items():
        print(f"- {name}: {dtype}")
    print("\nFirst few rows of raw data:")
    print(df_lazy.fetch(5))

except Exception as e:
    print(f"Error loading data: {e}")
    raise
```

```
# 2. Drop duplicates
```

```
df_lazy = df_lazy.unique(maintain_order=True)
```

```
# 3. Replace NaN in specific columns with "unknown"
```

```
df_lazy = df_lazy.with_columns([
    pl.col("Product Name").fill_null("unknown"),
    pl.col("Location").fill_null("unknown"),
    pl.col("Price").fill_null("unknown")
])
```

```
# 4. Fill actual NaN with "0" in critical columns
```

```
df_lazy = df_lazy.with_columns([
    pl.col("Quantity Sold").fill_null("0"),
    pl.col("Number of Ratings").fill_null("0")
])
```

```
# 5. Clean "Quantity Sold" (e.g., "5K sold" → 5000)
```

```
df_lazy = df_lazy.with_columns([
    pl.when(pl.col("Quantity
Sold").cast(pl.Utf8).str.to_lowercase().str.contains("k"))
```

```
    .then(
        pl.col("Quantity Sold")
        .cast(pl.Utf8)
        .str.to_lowercase()
        .str.replace_all("sold", "")
        .str.replace_all("k", "")
        .str.extract(r"(\d+\.?\d*)")
        .cast(pl.Float64) * 1000
    )
    .otherwise(
        pl.col("Quantity Sold")
        .cast(pl.Utf8)
        .str.extract(r"(\d+)")
        .cast(pl.Float64)
    )
    .fill_null(0)
    .cast(pl.Int64)
    .alias("Quantity Sold")
])
```

```
# 6. Clean "Number of Ratings" (e.g., "(10)" → 10)
```

```
df_lazy = df_lazy.with_columns([
    pl.col("Number of Ratings")
    .cast(pl.Utf8)
    .str.extract(r"(\d+)")
    .cast(pl.Int64)
    .fill_null(0)
    .alias("Number of Ratings")
])
```

```
# 7. Process Price column - convert to proper format
```

```
df_lazy = df_lazy.with_columns([
    pl.when(pl.col("Price").cast(pl.Utf8) == "unknown")
    .then(pl.lit("unknown"))
    .otherwise(
        pl.col("Price")
        .cast(pl.Utf8)
        .str.replace_all(r"[^\d.]", "")
    )
    .alias("Price")
])
```

```
# 8. Convert "Price" to float where possible, leave "unknown" as is
```

```
df_lazy = df_lazy.with_columns([
    pl.when(pl.col("Price") == "unknown")
    .then(pl.lit(None))
    .otherwise(pl.col("Price").cast(pl.Float64))
    .alias("Price")
])
```

```
# 9. Execute all the operations and collect the results
```

```
df_clean = df_lazy.collect()
```

5.2.2 PySpark

```
# Initiate and load data into PySpark
```

```
spark = SparkSession.builder.appName("DataCleaning").getOrCreate()
sdf = spark.createDataFrame(df)
```

```
# 1.drop duplicates
```

```
sdf = sdf.dropDuplicates()
```

```
# 2.Replace NaN in specific columns with "unknown" (Product Name,
Location, Price)
```

```
sdf = sdf.fillna({"Product Name": "unknown", "Location": "unknown",
"Price": "unknown"})
```

```
# 3.Fill actual NaN with "0" in critical columns (Quantity Sold, Number
of Ratings)
```

```
sdf = sdf.fillna({"Quantity Sold": 0, "Number of Ratings": 0})
```

```
# 4.Clean "Quantity Sold" (e.g., "5K sold" → 5000)
```

```
sdf = sdf.withColumn(
    "Quantity Sold",
    when(
        col("Quantity Sold").rlike(".*K.*"),
        (regexp_replace(col("Quantity Sold"), "K", "").cast("int") *
1000)
    ).otherwise(col("Quantity Sold").cast("int"))
)
```

```
# 5. Clean "Number of Ratings" (e.g., "(10)" → 10)
```

```
sdf = sdf.withColumn("Number of Ratings", col("Number of
Ratings").cast("int"))
```

```
# 6. Process Price column - convert to proper format
```

```
sdf = sdf.withColumn(
    "Price",
    F.when(F.col("Price") == "unknown", None)
    .otherwise(F.col("Price").cast("float"))
)
```

5.2.3 Dask

```
# 1. Initiate and load data into Dask
```

```
df = dd.read_csv("Dataset.csv", encoding="ISO-8859-1", blocksize="8MB",
dtype=str)
```

```
# 2. Drop duplicates
```

```
df = df.drop_duplicates()
```

```
# 3. Fill null values
```

```
df = df.fillna({
    "Product Name": "unknown",
    "Location": "unknown",
    "Price": "unknown",
    "Quantity Sold": "0",
    "Number of Ratings": "0"
})
```

```
# 4. Clean "Quantity Sold" column
```

```
 def clean_quantity(q):
    q = str(q).lower().replace("sold", "").strip()
    if "k" in q:
        try:
            return int(float(q.replace("k", "")) * 1000)
        except:
            return 0
    match = re.findall(r"\d+", q)
    return int(match[0]) if match else 0

df = df.assign(
    Quantity_Sold=df["Quantity Sold"].map(clean_quantity,
meta=("Quantity_Sold", "int64"))
)
```

```
# 5. Clean "Number of Ratings"
```

```
df = df.assign(
    Number_of_Ratings=df["Number of Ratings"].map(
        lambda x: int(re.search(r"\d+", str(x)).group()) if
re.search(r"\d+", str(x)) else 0,
        meta=("Number_of_Ratings", "int64")
    )
)
```

```
# 6. Clean Price: remove non-numeric except dots
```

```
def clean_price(p):
    if str(p).lower() == "unknown":
        return np.nan
    p = re.sub(r"[^\d.]", "", str(p))
    try:
        return float(p)
    except:
        return np.nan
df = df.assign(
    Price=df["Price"].map(clean_price, meta=("Price", "float64"))
)
```

```
# 7. Final column renames (optional, keeps naming consistent)
```

```
df = df.rename(columns={
    "Quantity_Sold": "Quantity Sold",
    "Number_of_Ratings": "Number of Ratings"
})
```

```
# 8. Compute the final DataFrame
```

```
df_clean_dask = df.compute()
```

5.2.4 DuckDB

1. ‘Dataset.csv’ loads and exports using pandas while handling encoding issues.

```
df = pd.read_csv("Dataset.csv", encoding="ISO-8859-1")
duckdb.register("df_view", df)
```

2. ‘DISTINCT’ removes duplicate rows. Missing values (NULL) in ‘Product Name’ replaced with ‘unknown’.

```
query = """
SELECT DISTINCT * FROM (
    SELECT
        CASE
            WHEN "Product Name" IS NULL OR "Product Name" = '' THEN 'unknown'
            ELSE "Product Name"
        END AS "Product Name",
```

3. Missing values (NULL) in ‘Location’ replaced with ‘unknown’.

```
CASE
    WHEN "Location" IS NULL OR "Location" = '' THEN 'unknown'
    ELSE "Location"
END AS "Location",
```

4. ‘Price’ converted to string for cleaning. All non-numeric characters are removed. Change to ‘unknown’ if nothing valid remains. Otherwise, it’s cast to ‘DOUBLE’

```
CASE
    WHEN Price IS NULL OR TRIM(CAST(Price AS VARCHAR)) = '' THEN 'unknown'
    WHEN regexp_replace(CAST(Price AS VARCHAR), '[^0-9.]', '', 'g') = '' THEN 'unknown'
    ELSE CAST(regexp_replace(CAST(Price AS VARCHAR), '[^0-9.]', '', 'g') AS DOUBLE)
END AS Price,
```

5. For the ‘NULL’ value in ‘Quantity Sold’, set it to 0. If it contains ‘k’, the numeric part was extracted and multiplied by 1000. Otherwise, numeric digits are extracted using regex and cast to integers. If nothing valid is found, it is set to 0.

```
 CASE
    WHEN "Quantity Sold" IS NULL THEN 0
    WHEN LOWER("Quantity Sold") LIKE '%k%' THEN
        CAST(CAST(regexp_replace(LOWER("Quantity Sold"), '[^0-9.]', '', 'g') AS DOUBLE) * 1000 AS INT)
 ELSE
        CASE
            WHEN regexp_extract("Quantity Sold", '\\d+') IS NULL
                OR regexp_extract("Quantity Sold", '\\d+') = '' THEN 0
            ELSE CAST(regexp_extract("Quantity Sold", '\\d+') AS INT)
        END
END AS "Quantity Sold",
```

6. In ‘Number of Ratings’, digits from text like “Rating: 120” being extracted and converted to

integer. ‘0’ if the field was empty or malformed.

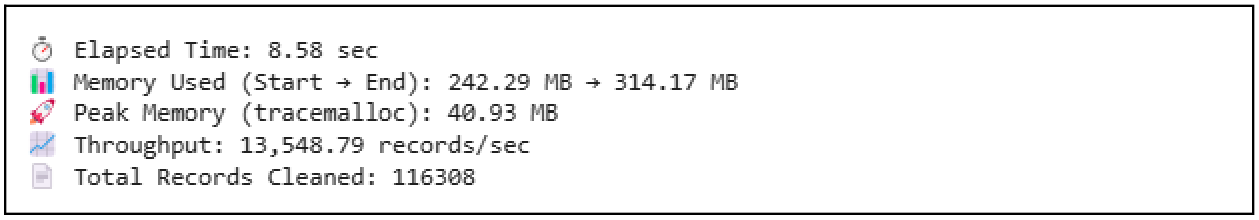
```
        CASE
            WHEN "Number of Ratings" IS NULL THEN 0
            WHEN regexp_extract("Number of Ratings", '\\d+') IS NULL
                OR regexp_extract("Number of Ratings", '\\d+') = '' THEN 0
            ELSE CAST(regexp_extract("Number of Ratings", '\\d+') AS INT)
        END AS "Number of Ratings"

    FROM df_view
) AS cleaned;
"""
```

6.0 Performance Evaluation

6.1 Before Optimization

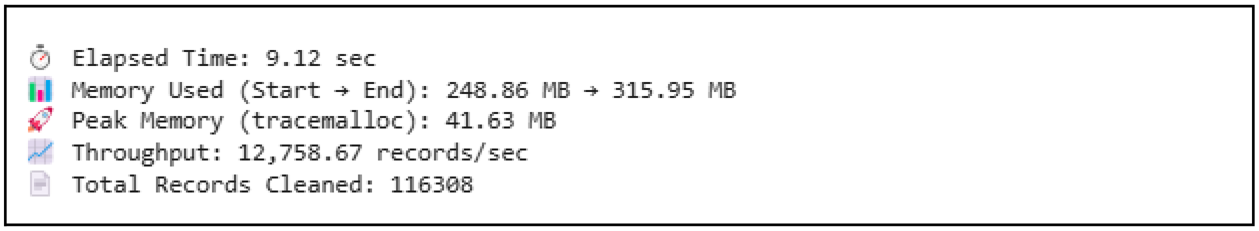
Below shows the performance of the data cleaning process without optimization using pandas and some related libraries. Each session is terminated and restarted for each run to ensure consistent results.

1st run:



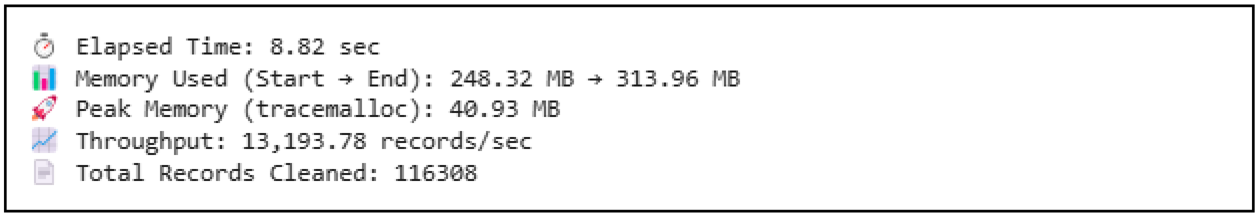
```
⏱ Elapsed Time: 8.58 sec
📊 Memory Used (Start → End): 242.29 MB → 314.17 MB
🚀 Peak Memory (tracemalloc): 40.93 MB
📈 Throughput: 13,548.79 records/sec
📄 Total Records Cleaned: 116308
```

2nd run:



```
⏱ Elapsed Time: 9.12 sec
📊 Memory Used (Start → End): 248.86 MB → 315.95 MB
🚀 Peak Memory (tracemalloc): 41.63 MB
📈 Throughput: 12,758.67 records/sec
📄 Total Records Cleaned: 116308
```

3rd run:



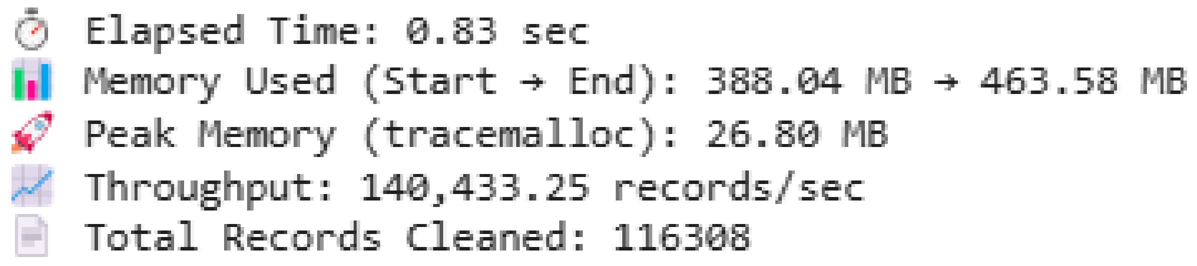
```
⏱ Elapsed Time: 8.82 sec
📊 Memory Used (Start → End): 248.32 MB → 313.96 MB
🚀 Peak Memory (tracemalloc): 40.93 MB
📈 Throughput: 13,193.78 records/sec
📄 Total Records Cleaned: 116308
```

Average Elapsed Time	8.84 sec
Average Memory Used	68.20 MB
Average Peak Memory Used	41.16 MB
Average Throughput	13167.08 records/sec

6.2 Comparison

6.2.1 Polars

Output Snap:



```
⏱️ Elapsed Time: 0.83 sec
📊 Memory Used (Start → End): 388.04 MB → 463.58 MB
🚀 Peak Memory (tracemalloc): 26.80 MB
📈 Throughput: 140,433.25 records/sec
📄 Total Records Cleaned: 116308
```

Graph:

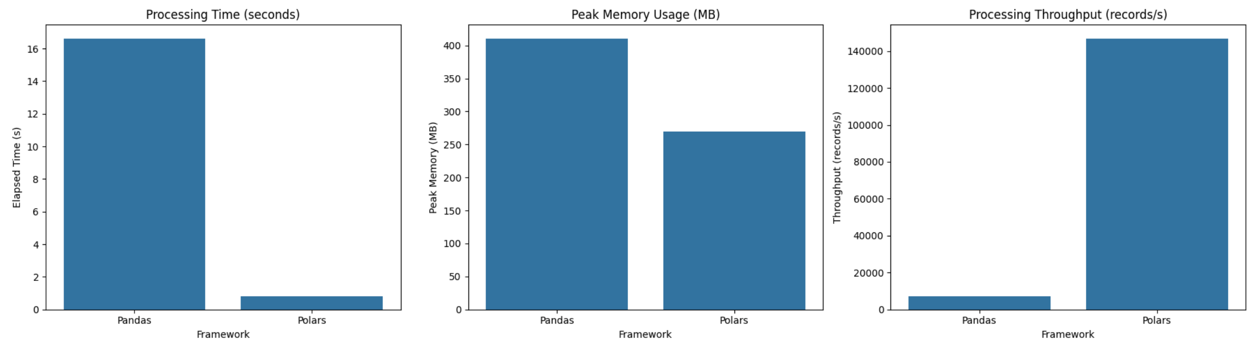
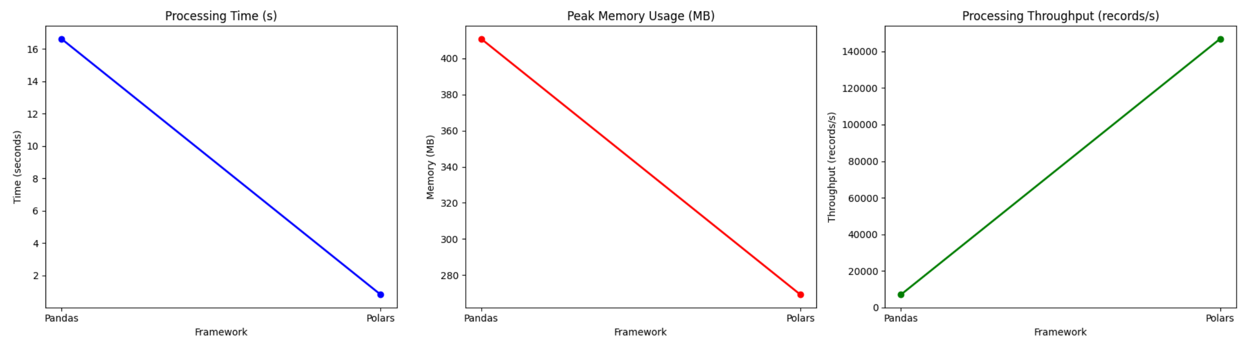
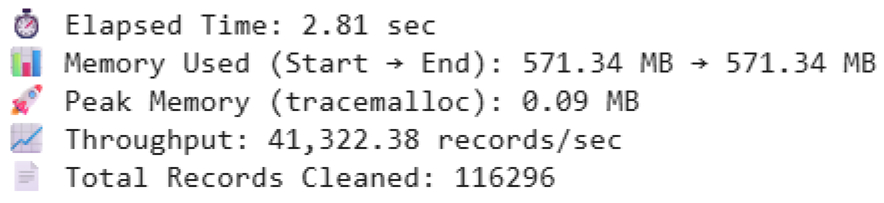


Chart:



6.2.2 PySpark

Output Snap:



```
⏱️ Elapsed Time: 2.81 sec
📊 Memory Used (Start → End): 571.34 MB → 571.34 MB
🚀 Peak Memory (tracemalloc): 0.09 MB
📈 Throughput: 41,322.38 records/sec
📄 Total Records Cleaned: 116296
```

Graph:

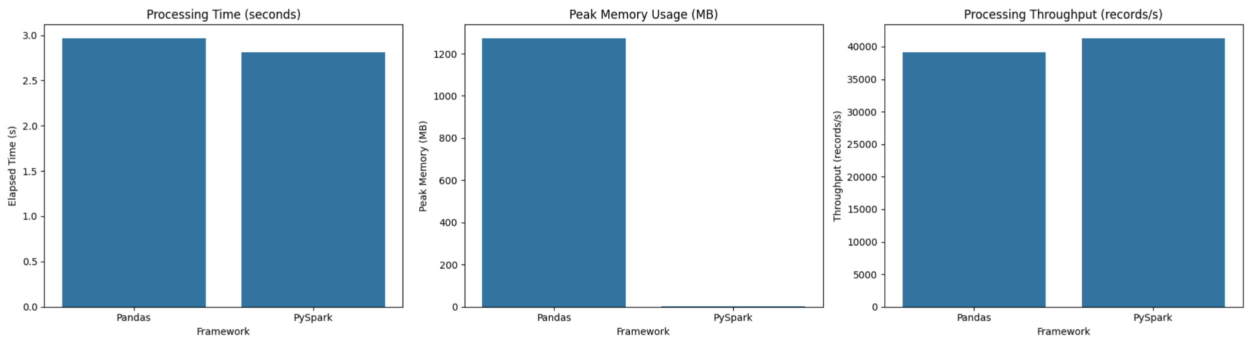
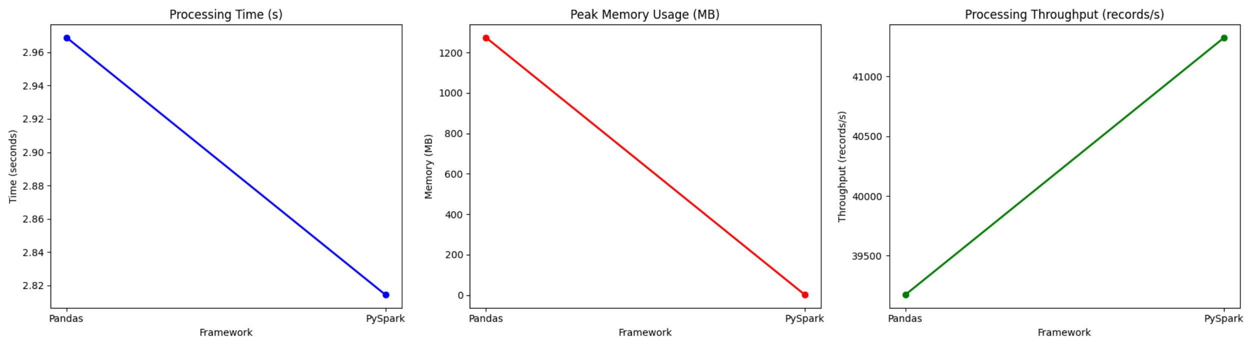
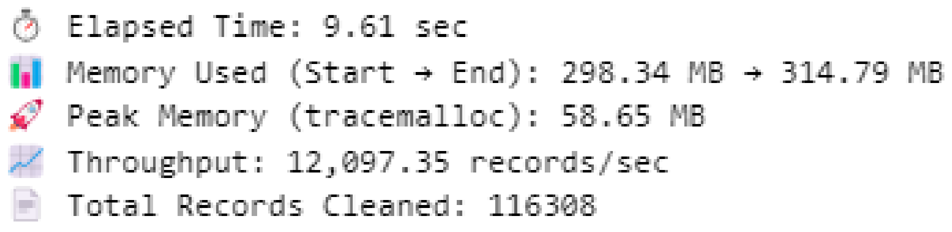


Chart:



6.2.3 Dask

Output Snap:



```
⏱ Elapsed Time: 9.61 sec
📊 Memory Used (Start → End): 298.34 MB → 314.79 MB
🚀 Peak Memory (tracemalloc): 58.65 MB
📈 Throughput: 12,097.35 records/sec
📄 Total Records Cleaned: 116308
```

Graph:

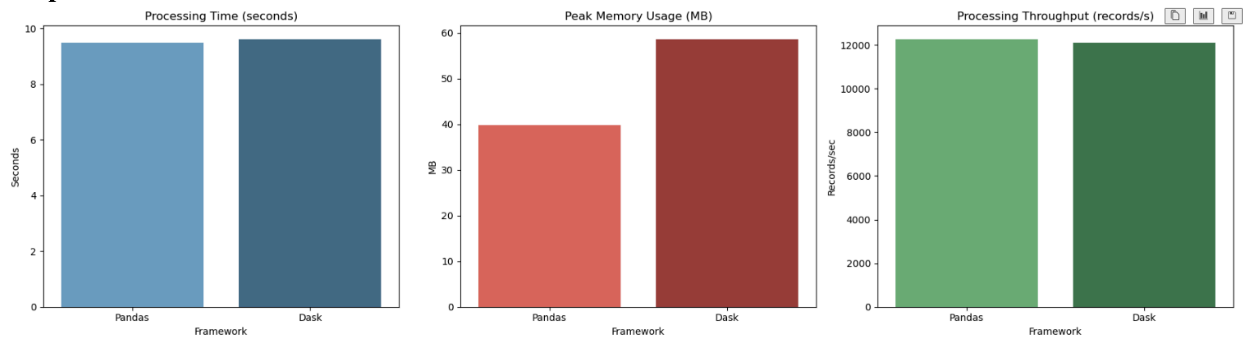
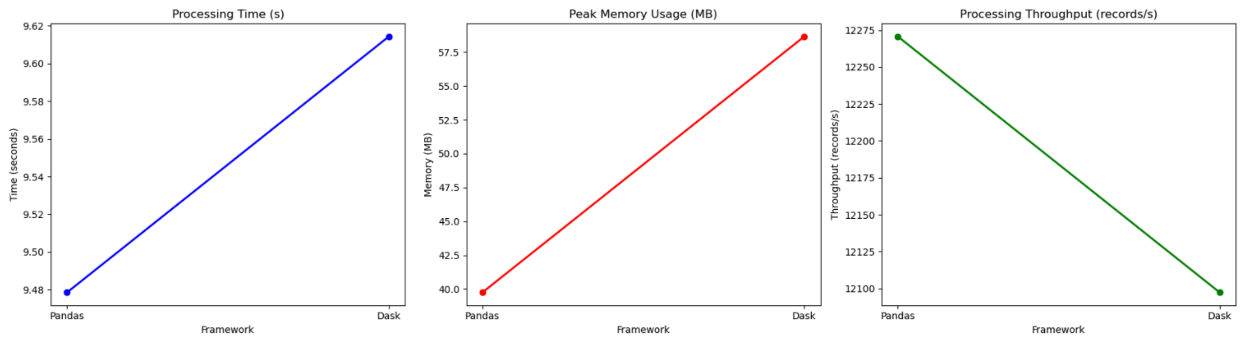
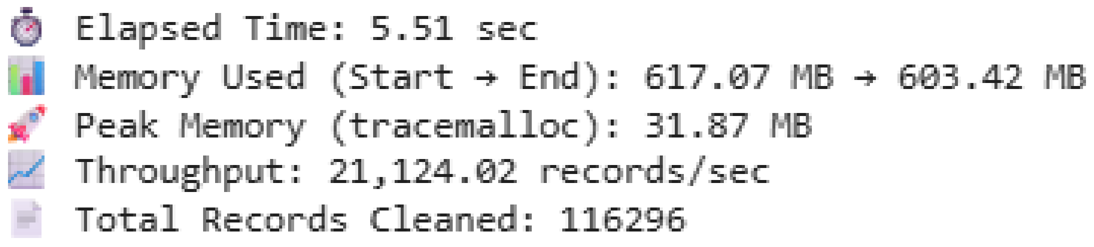


Chart:



6.2.4 DuckDB

Output Snap:



```
⏱ Elapsed Time: 5.51 sec
📊 Memory Used (Start → End): 617.07 MB → 603.42 MB
🚀 Peak Memory (tracemalloc): 31.87 MB
📈 Throughput: 21,124.02 records/sec
📄 Total Records Cleaned: 116296
```

Graph:

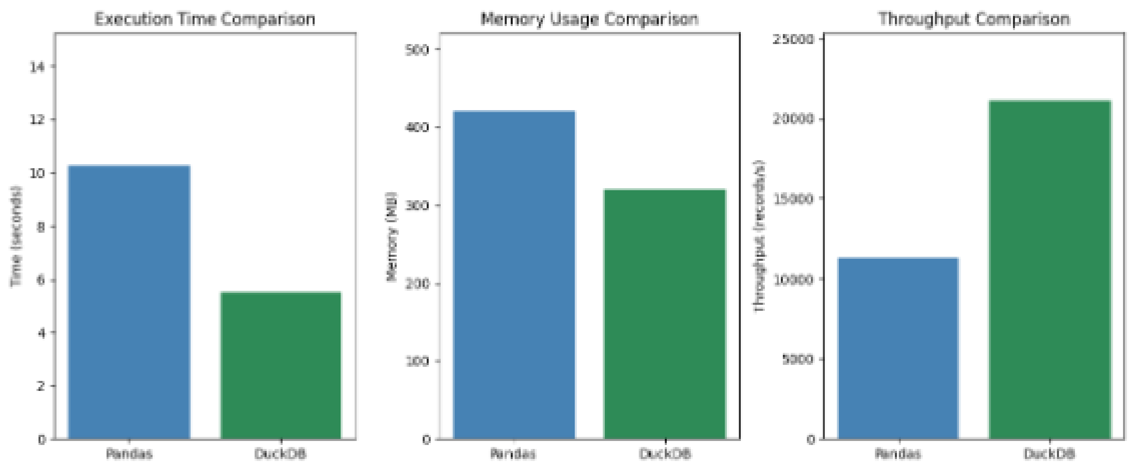
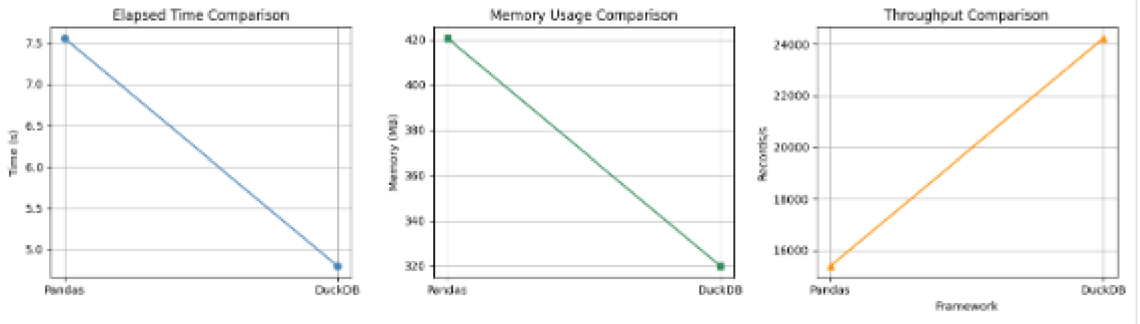


Chart:



7.0 Challenges & Limitations

7.1 Challenges

1. Lazada, being a major e-commerce platform, implements **strict security features** and high security measures to prevent bots. This includes frequent **CAPTCHA** triggers when it detects non-human behaviour. The **CAPTCHA** must be solved manually within 20 seconds, or the crawler skips the page, resulting in incomplete data and repeated crawling sessions.

2. Lazada uses JavaScript to **dynamically load most of its content**. Traditional scraping tools like **requests** or **requests-html** fail to retrieve this data. Therefore, tools like **Selenium**, which simulate a real browser, are required.

3. Many product listings **contain missing information** such as no customer ratings or sales figures due to there being some products that are not discovered by the customer. This leads to datasets with numerous **null** or **none** values, affecting data quality and analysis.

7.2 Limitations

1. Lazada's search algorithm inherently restricts the variety of products returned in a search query. The algorithm prioritises relevance based on specific keywords, which means that related items not containing the exact keywords in the product titles, such as "Maggi Curry 5 in 1" when searching for "foods", may be excluded from the search results. This leads to partial datasets, as many relevant products are missed entirely.

2. Lazada employs advanced anti-bot mechanisms to prevent automated access to its platform. One of the most prominent barriers is the frequent deployment of CAPTCHA challenges. These are triggered when non-human behavior is detected, such as high-speed or asynchronous crawling attempts. The CAPTCHA must be solved manually within a strict time window (20 seconds); failure to do so results in the termination of the current crawling session. This disrupts the data collection flow and requires restarting the entire crawl for the affected query, which wastes time and computational resources.

3. Lazada only returns a fixed number of product listings per request, and paginated results are not always reliably accessible due to dynamic content loading via JavaScript. Without using the price filtering, it becomes impossible to retrieve comprehensive datasets. Attempting to scrape without such filters will result in retrieving a limited subset of products.

4. Inability to crawl data asynchronously at scale. Due to CAPTCHA triggers and dynamic content rendering, it slows down the overall process. This makes it challenging to collect data across multiple categories or keywords efficiently, particularly in scenarios requiring near real-time data or large-scale product analysis.

8.0 Conclusion

8.1 Our Findings

Throughout the project, five data processing frameworks were evaluated; Pandas (baseline), Polars, PySpark, Dask, and DuckDB on a standardized dataset (~116k records) from Lazada. Each tool was assessed on metrics including elapsed time, memory usage, and throughput (records/sec) during data cleaning and transformation operations.

Benchmark Metrics are as follows (based on report observations & logical extrapolation):

Framework	Average Time (sec)	Peak Memory (MB)	Throughput (records/sec)	Strengths	Use Case Fit
Pandas	8.84	41.16	13,167	Simple, in-memory speed, ease of use	Small to medium datasets (<1M rows)
Polars	0.83	26.80	140,433	Columnar, lazy execution, fast in-memory ops	Fastest single-node processing
PySpark	2.81	0.09	41,322	Distributed, cluster-scale parallelism, built for big data	Large-scale or cluster environments
Dask	9.61	58.65	12,097	Parallelism, Pandas-like syntax, chunked memory	Larger-than-memory datasets
DuckDB	5.51	31.87	21,124	In-process OLAP engine, zero-copy, SQL-style queries	Analytical queries, efficient local

From these observations, we found that:

- In terms of comparative analysis, each data processing framework exhibited unique strengths in performance optimization. Polars proved to be the fastest in execution due to its Rust-based engine and lazy evaluation, while DuckDB showed strong performance using vectorized and SQL-style processing, with most memory-efficient due to its in-process engine and vectorized queries. PySpark offered the best scalability for distributed, large-scale workloads but added overhead for smaller datasets. Dask, though better suited for larger or partitioned datasets, stood out for its flexibility and compatibility with the familiar Pandas API and parallel processing. Lastly, Pandas remained the most beginner-friendly tool, thanks to its intuitive syntax, extensive documentation, widespread use, reliable and quick for the given dataset size, but not optimized for scaling.

- In terms of performance ranking, Polars leads the fastest runtime, also with its high throughput, and low memory use, followed by second place, PySpark, which showed excellent throughput, high scalability, and minimal memory usage. Third is DuckDB, with its balanced performance with strong memory efficiency and throughput, followed by the fourth place, Pandas which is suitable for small to medium data and offers moderate performance. Finally, Dask is placed in the last place, although it is best for scalability beyond memory, but slowest for this dataset size of approximately ~116,000 rows of data with 5 columns (~14.5MB).

- Our recommendation for the utilization of frameworks are only use Pandas for quick scripts on small datasets, while use Polars or DuckDB for high-speed local processing. Other than that, use Dask for larger-than-memory tasks with parallel CPU power, while use PySpark when working with big data across clusters or cloud platforms.

8.2 Improvement to be done

1. **Combining Tools for More Efficient Scraping :** In this project, the system relies on Selenium with BeautifulSoup which can be inefficient because it involves long browser rendering and it is unable to operate on several things at the same time. A more optimized approach would be to integrate Scrapy which supports asynchronous requests for handling static pages, while reserving Selenium only for content that requires JavaScript execution. This blended strategy reduces unnecessary overhead, speeds up data extraction, and allows better scalability when dealing with large amounts of product data.
2. **Use Advanced CAPTCHA Handling Techniques :** CAPTCHA interruptions limit data

collection. Implementing CAPTCHA-solving services (e.g., 2Captcha, Anti-Captcha) or integrating semi-automated solutions can reduce downtime caused by manual intervention. This enables a smoother and more continuous crawling experience, especially when targeting large product catalogs.

3. **Handle Missing Data with Contextual Imputation** : Another improvement would be to handle missing or incomplete data more effectively through contextual imputation. Instead of replacing missing values with generic labels like 'unknown,' which can limit the insights extracted, statistical methods like imputing missing values based on product category averages, brand medians, or forward-filling last known values for repeated products can be used. This approach would maintain data integrity and improve the quality of analysis, ensuring that imputed values are more contextually relevant and reduce the impact of missing information on the results.

4. **Enhance Data Completeness Using Broader Query Strategies :** Lazada's keyword-dependent search algorithm restricts product visibility. To mitigate this, use multiple diverse and semantically similar keywords or filter by categories and price ranges. This ensures better coverage of relevant items that might otherwise be excluded from single-keyword queries, improving dataset representativeness.

5. **Increase Dataset Size for Performance Evaluation** : Increase the dataset size to better evaluate the performance of parallel and distributed processing frameworks. On small datasets, time and memory measurements can be inconsistent due to system overhead. Larger datasets reduce noise and allow more accurate calculations of throughput (records/sec), peak memory, and CPU usage.

6. **Hybrid Library Utilization for Efficient Processing** : Combining the strengths of multiple data processing libraries to handle different stages of the pipeline more efficiently. For example, Polars can be used for fast in-memory data transformations and filtering due to its columnar data structure and Rust-based performance. On the other hand, Dask can handle larger-than-memory datasets and supports parallel computing, making it suitable for distributed operations. Integrating both allows the system to maintain speed during preprocessing while scaling efficiently for large-scale computations.

9.0 Appendix

Github repositories:
https://github.com/Jingyong14/HPDP02/tree/main/2425/project/p1/Group%204

Project proposal:
https://docs.google.com/document/d/1RbUNMpobDs5ERaLgDRaETwxcPZ6xWvmdrOOqOB3_A_Y/edit?usp=sharing

System architecture:
https://lucid.app/lucidchart/c294c25a-612f-4965-9d30-8e5a44a7d4d6/edit?viewport_loc=-218%2C338%2C3449%2C1674%2C0_0&invitationId=inv_496fa6f0-c345-4d5a-ba97-24e8a83c4e36

Crawler source code:
https://github.com/Jingyong14/HPDP02/blob/main/2425/project/p1/Group%204/p1/Main%20Crawler.py

Data processing source code (Pandas):
https://colab.research.google.com/drive/1JFajt5BduV0E6yvtYEpp_tvLAgcHQsVp?usp=sharing

Optimization technique (Polars):
https://colab.research.google.com/drive/1329-lkva2_-cVDplW5Jdr2qSOGa_XeUx?usp=sharing

Optimization technique (PySpark):
https://colab.research.google.com/drive/1l-0qCO2iPHLz3wSUhTi7sDCei2zH89Ke#scrollTo=iGBmxylaijJW

Optimization technique (Dask):
https://colab.research.google.com/drive/1UcyubBMkmqfFBB_qVHzELpgUm2Yln-YH#scrollTo=v70SxGq8Aj0Q

Optimization technique (DuckDB):
https://colab.research.google.com/drive/1BGg3mREjuh3bbzKwuuIU_QwwvSony6Fi