HPDP - PROJECT 1 REPORT.pdf

by Wan Nur Sofea Mohd Hasbullah

Submission date: 15-May-2025 01:37AM (UTC-0700)
Submission ID: 2676461617
File name: HPDP_-_PROJECT_1_REPORT.pdf (2.22M)
Word count: 3924
Character count: 20411

SECP3133 HIGH PERFORMANCE DATA PROCESSING

SEMESTER 2 2024/2025

Project 1

Title: Optimizing High-Performance Data Processing for Large-Scale Web Crawlers – PG Mall

SECTION	01
GROUP	D
GROUP MEMBERS	1. WAN NUR SOFEA BINTI MOHD HASBULLAH (A22EC0115) 2. LOW YING XI (A22EC0187) 3. MUHAMMAD ARIFF DANISH BIN HASHNAN (A22EC0204) 4. MUHAMMAD IMAN FIRDAUS BIN BAHARUDDIN (A22EC0216)
LECTURER	ASSOC. PROF. DR. MOHD SHAHIZAN BIN OTHMAN
SUBMISSION DATE	15 May 2025

TABLE OF CONTENT

1.0 INTRODUCTION

This section covers the background and objectives of this project. It describes the targeted website and specific data fields to be collected.

1.1 BACKGROUND

Online shopping platforms like Amazon, Taobao and Shopee generate a large amount of semi-structured and unstructured data like product links, image URLs and price range. An efficient and reliable data extraction and processing system is crucial to gain valuable insights, drive decision-making and maintain a competitive advantage. The challenge in this kind of system is to achieve significant computational efficiency due to the data scale and diversity.

This project finds a way to build a scalable system that is capable of extracting, cleaning and analysing real-world data to address the need for high-performance data processing. This system emphasises the application of web crawling, data preprocessing and evaluating different text-processing libraries under several performance constraints.

1.2 OBJECTIVES

The objectives of this project are below:

1. Build a web crawler that extracts a minimum of 100,000 data from the PG Mall website in an automated and scalable manner.
2. Perform text processing and data cleaning on the scraped data using effective text normalisation and noise removal techniques.
3. Compare text processing libraries (Pandas, Dask and Polars) based on key performance indicators such as processing time, memory usage, CPU usage and throughput.
4. Apply high-performance computing techniques to optimise the text processing pipeline.
5. Evaluate and document the system's performance, identifying bottlenecks and discussing improvements made.

1.3 TARGET WEBSITE AND DATA

The targeted website of this project is PG Mall website. The figure below shows the snapshot of the PG Mall website accessed through desktop.

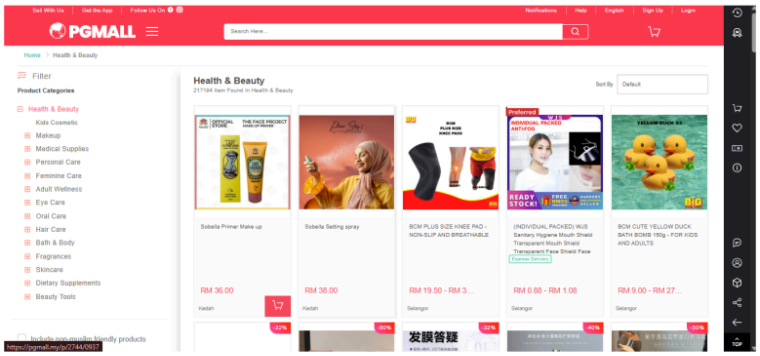

Figure 1.3.1 Snapshot of PG Mall website

The data extracted is related to the product category 'Health & Beauty'. The fields involved are product name, product type, link, price and location. From the webpage, the product name, price and location of a certain product can be observed, as shown in Figure 1.3.2. The link of each product is extracted from the HTML of the webpage. Plus, the product type of each item is obtained by extracting data from each subcategory (refer to the left sidebar in Figure 1.3.1) and mapping it to the dataset by comparing the link of the product. Table 1.3.1 shows the data field that will be included in the raw dataset.

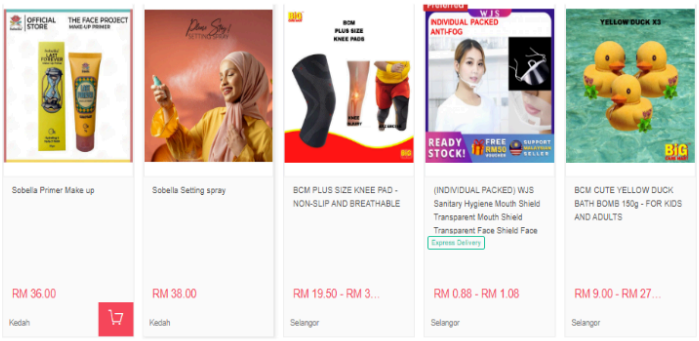

Figure 1.3.2 Snapshot of Section That Includes Data To Be Extracted

Table 1.3.1 Data Field and Description

Data field	Description
product_name	Name of the health and beauty item sold on the website
link	URL link of each item sold
price	Cost of item sold in Malaysian Ringgit (RM)
location	Location where the item is sold or shipped from
product_type	Category of item (Example: Skincare, Eye Care, Medical Supplies)

2.0 SYSTEM DESIGN & ARCHITECTURE

This section introduces the architecture of the project, tool and framework to develop the project and the role of each member of the team.

2.1 SYSTEM ARCHITECTURE

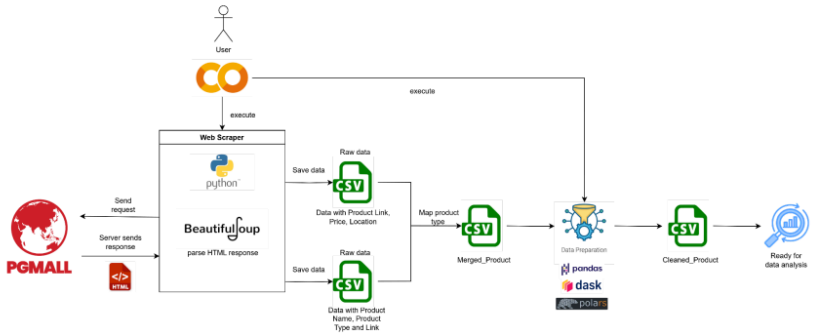

Figure 2.1.1 Architecture Diagram

Figure 2.1.1 is the architecture diagram that illustrates the workflow of the web scraping and data processing pipeline used in this project. The process starts with the user initiating the execution of Python script for web scraping through Google Colab. Google Colab serves as the development and runtime environment for this project.

The web scraper is developed using Python and libraries like requests and Beautiful Soup. The web scraper sends HTTP requests to PG Mall website, which is the data source for this project. The server of PG Mall responds with HTML content. The HTML content is then parsed and extracted by the web scraper.

The extracted data is stored as a CSV file. There are two types of raw data extracted from PG Mall website: one from the category 'Health & Beauty' and another one from all the subcategories of 'Health & Beauty'. This step ensures the raw data is complete with the product type to provide more meaningful insights during data analysis. After obtaining both types of raw data, the data is mapped to the product type by comparing the link of the product. The following

step is the data cleaning process to remove duplicates, replace null values and perform text processing as well as standardize the dataset for further analysis. The final output is the dataset after data cleaning and processing and is ready for further analysis.

This architecture serves as the foundation to be optimized using different techniques for the scraping process and data processing in the following sections.

2.2 TOOLS AND FRAMEWORK USED

This project applied diverse tools and frameworks, as shown in Table 2.2.1. For the web scraping process, the code was written in Python using libraries such as Beautiful Soup and Request. The Pandas library was used for the data cleaning process. Dask and Polars were applied for optimization to improve the processing performance. All the codes were executed in Google Colab.

Table 2.2.1 Tools and Frameworks Used

Tools/Framework	Description
Beautiful Soup	Python library for parsing HTML and extracting data from web pages
Request	HTTP library for Python to fetch web pages
Python	Programming language used to write web scraping scripts
Google Colab	Cloud-based Jupyter notebook environment for scraping scripts
Pandas	Data manipulation library that is used for data cleaning and data analysis
Dask	Parallel computing library that extends pandas for larger volume data
Polars	Dataframe library designed for fast and memory-efficient data manipulation.

2.3 ROLE OF TEAM MEMBERS

This section describes the roles and tasks of each member in the team.

Table 2.3.1 Role of Team Members

Name	Role	Task Description
WAN NUR SOFEA	Group Leader	- Choose a Malaysian website - Coordinated group tasks - Develop scraping of items by subcategory - Test data cleaning
LOW YING XI	Architect, Evaluator	- Design system architecture - Develop mapping phase of raw data - Handle performance evaluation and result compilation
ARIFF DANISH	Coder	- Develop and test optimized web crawler - Ensure accurate data extraction - Develop optimized data cleaning
IMAN FIRDAUS	Coder	- Develop and test web crawler - Ensure data completeness and consistency - Develop data cleaning

3.0 DATA COLLECTION

The data collection phase gathers several descriptions found on each product. The data is saved in a CSV file named “Item_list.csv”. Excluding duplicates, there are several noises found in the initial data as below.

Data field	Initial Status
product_name	Contain characters outside of the normal ASCII
link	No problem
price	String type instead of float, has a range of prices and string “RM” in cells
location	Contain null cells
product_type	Contain null cells

3.1 CRAWLING METHOD

This project applied Requests library to deliver HTTP requests to the site and employed the BeautifulSoup library to parse and extract relevant product data from the HTML structure received from site’s response. There are a few methods used when scraping the web:

- Pagination
  - Uses sequential pagination via *while* loop in main() and incrementing the page number until no data is returned (empty response) or data limit reached.

- Rate-Limiting
  - Retry logic for HTTP 429 error (too many requests) which waits for 5 seconds between retries. 3 retries per page.
  - Concurrency control which uses *ThreadPoolExecutor* with max_workers of 5 to limit parallel requests.

- Asynch
  - Multithreaded implementation via *concurrent.futures.ThreadPoolExecutor*

3.2 NUMBER OF RECORDS COLLECTED

The number of records collected is approximately the same as the limit is set to 200000 data in the source code in the variable *max_data_limit* to reduce time taken for scraping and also resources.

Optimization	Numbers of Data Collected
No optimization	200009
Multithreading	200010

The code also applied other termination conditions as below:

- A page returns no data
- *total _data_saved >= max_data_limit*

3.3 ETHICAL CONSIDERATIONS

The website does not have any restrictions related to server load. The proof is the application of zero delay on requests. However, a few considerations are still taken and integrated into the scraping process (source code) to adhere to the ethical considerations .

- Respects *Retry-After* headers during rate limiting
- Uses a generic *User-Agent* string to mimic a browser.

The website's "robots.txt" at https://pgmall.my/robots.txt has also been manually checked and is allowed. However, to confirm it, a few lines of code were run and the result was shown as below.

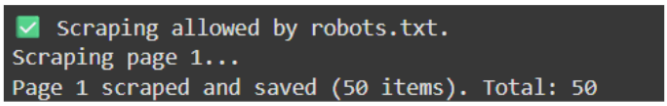

Figure 3.3.1 "robots.txt" Result Output

4.0 DATA PROCESSING

This section describes the data processing steps used to map, clean, transform and standardize the raw data extracted by the web scraper into a structured and standardized dataset ready for further analysis.

4.1 MAPPING PHASE

As shown in the architecture diagram from the previous section, Section 2.0, the raw data underwent a mapping process to map the data field 'product type'. For the mapping phase, the process started with the merge of the items from each subcategory of ‘Health & Beauty’, as shown in Figure 4.1.1. The input files are product items from the subcategories of ‘Health & Beauty’. Each input file represents products from one subcategory. The output file is “merged_product_list.csv”, which has three data fields: product name, product type and link of the product.

```python
# Define the output file
output_file = "merged_product_list.csv"

# Automatically list all files in the working directory
csv_files = [f for f in os.listdir() if f.startswith("Item_list_") and f.endswith(".csv")]

merged_data = []

for file in csv_files:
    # Extract product type from filename
    product_type = file.replace("Item_list_", "").replace(".csv", "").strip()

    # Load CSV data
    df = pd.read_csv(file)

    # Drop duplicates within this file based on 'product_name' and 'link'
    df.drop_duplicates(subset=['product_name', 'link'], inplace=True)

    # Add the product_type column
    df['product_type'] = product_type

    merged_data.append(df)

# Concatenate all DataFrames
final_df = pd.concat(merged_data, ignore_index=True)

# Save to a new CSV
final_df.to_csv(output_file, index=False, encoding='utf-8-sig')

print(f"Merged file saved as: {output_file}")
files.download(output_file)
```

```
Merged file saved as: merged_product_list.csv
```

Figure 4.1.1 Snapshot of Code in "Map_ProductType.ipynb"

After that, the process continued with the mapping steps between the "merged_product_list.csv" and "Item_list.csv". "Item_list.csv" is the output of the scraping process that consists of data from the category of 'Health & Beauty' with data fields like product name, link, price, and location but lacking the product type. These two files were then mapped together to get the output file "updated_item_list.csv", as shown in Figure 4.1.2. "updated_item_list.csv" consists of 5 data fields: product name, price, location, link, and product type.

```python
# Load the merged file that contains 'link' and 'product_type'
merged_file = "merged_product_list.csv"
merged_df = pd.read_csv(merged_file)

# Create a mapping from link to product_type
link_to_type = dict(zip(merged_df['link'], merged_df['product_type']))

# Load original file to update
original_file = "Item_list.csv"
df = pd.read_csv(original_file)

# Map the product_type from the merged file based on the link
df['product_type'] = df['link'].map(link_to_type)

# Save the updated file (overwrite or create a new one)
updated_file = "updated_item_list.csv"
df.to_csv(updated_file, index=False, encoding='utf-8-sig')
print(f"Updated: {updated_file}")


files.download(updated_file)
```

```
Updated: updated_item_list.csv
```

Figure 4.1.2 Snapshot of Code in "Map_ProductType.ipynb"

4.2 DATA CLEANING

After obtaining all the data fields needed, the raw data in the file "updated_item_list.csv" underwent a cleaning process. The dataset was loaded into the Python script using the Pandas library and checked for duplicated data and null values, as shown in Figure 4.2.1.

```
import pandas as pd
import re
import time
import psutil
import os

# Load dataset
df = pd.read_csv("updated_item_list.csv")

# Log initial shape
initial_shape = df.shape
```

```
print("Duplicated data:",(df.duplicated()).sum())


# Check for nulls by column (log or print if needed)
null_counts = df.isnull().sum()
print("Null values by column:\n", null_counts)
```

```
Duplicated data: 7
Null values by column:
 product_name       0
link                0
price               0
location         2947
timestamp           0
product_type      432
dtype: int64
```

Figure 4.2.1 Snapshot of Code in "Cleaning_Process_withoutOptimization.ipynb"

From Figure 4.2.1, there were 7 duplicated data and two data fields, which are location and product type had null values. Thus, the duplicated data was dropped and since these two fields are string-type data, the null values were replaced with "Unknown" to categorize the missing or unknown data into the same group for further analysis, as shown in Figure 4.2.2.

```
# Drop duplicate rows
df = df.drop_duplicates()

# Fill in null
df['location'].fillna('Unknown', inplace=True)
df['product_type'].fillna('Unknown', inplace=True)
```

Figure 4.2.2 Snapshot of Code in "Cleaning_Process_withoutOptimization.ipynb"

é´²è„±å'æ´—å'æ°´/anti hairloss shampoo/ç"Ÿå'å¢žå'ŽßæºÅŽ»å±'æ-	https://pgmall.my/p/2737/	RM40.00	Sar
10pcs 2 way Medical Silicone Foley Catheter Urethral Sounds tube Men's	https://pgmall.my/p/2737/	RM419.86	Mai
Medical PVC Manual Resuscitation Adult First Aid Kit Ambulance Config	https://pgmall.my/p/2737/	RM345.52	Mai
Pet dog excrement remover for cleaning excrement pet outdoor cleaning	https://pgmall.my/p/2737/	RM 334.18 - RM 422.24	Mai
Portable Large Capacity Urine Catcher Elderly Patients Bedridden with Li	https://pgmall.my/p/2737/	RM332.92	Mai

Figure 4.2.3 Snapshot of Raw Data in "updated_item_list.csv "

From Figure 4.2.3, the raw data was observed to contain some unreadable characters in the product names and some product prices were extracted as ranges and stored as strings with "RM" at the beginning. To handle the issue, a function called "extract_lowest_price" was designed, as shown in Figure 4.2.4, to extract the numeric value. For price ranges, the lowest value in the range was extracted as the item's price. The function was called, as shown in Figure 4.2.5 and the result was rounded to 2 decimal places. The standardized prices were stored in the new column called "cleaned_price" and the original price column was dropped. The product name was cleaned by removing the unreadable character, as shown in Figure 4.2.5.

```
# Clean price column and extract lowest numeric value
def extract_lowest_price(price_str):
    if pd.isna(price_str):
        return None
    matches = re.findall(r"\d+(?:\.\d+)?", price_str)
    if matches:
        return float(matches[0])
    return None
```

Figure 4.2.4 Snapshot of Code in "Cleaning_Process_withoutOptimization.ipynb"

```
# Standardize price format
df['cleaned_price'] = df['price'].apply(extract_lowest_price)
df['cleaned_price'] = df['cleaned_price'].round(2)

df.drop(columns=['price'], inplace=True)

# Clean unreadable characters from product_name
df['product_name'] = df['product_name'].apply(lambda x: re.sub(r"[^\x00-\x7F]+", '', str(x)))
```

Figure 4.2.5 Snapshot of Code in "Cleaning_Process_withoutOptimization.ipynb"

After completing all the cleaning processes, the data fields in the dataset were renamed, as shown in Figure 4.2.6. Then, all the data was saved into a .csv file called "Item_list_cleaned.csv".

```
df.rename(columns={
    'product_name': 'Product Name',
    'cleaned_price': 'Price',
    'location': 'Location',
    'link': 'Link',
    'product_type': 'Product Type'
}, inplace=True)

# Save cleaned data
df.to_csv("Item_list_cleaned.csv", index=False, float_format='%.2f')
```

Figure 4.2.6 Snapshot of Code in "Cleaning_Process_withoutOptimization.ipynb"

4.3 DATA STRUCTURE

Figure 4.3.1 is the final dataset after the data cleaning process. Observing that the product name is clean without unreadable characters and the price is converted to float type in the correct format.

Product Name	Link	Location	Product Type	Price
MuscleRulz L-Carnitine 3000mg (33 Servings) (READ DES	https://pgmall.my/p/2740/8004	Selangor	Unknown	128
MuscleRulz Iso Rulz (5LBS)	https://pgmall.my/p/2740/6329	Selangor	Unknown	314
Kevin Levrone Gold Whey (2kg)	https://pgmall.my/p/2740/6306	Selangor	Unknown	265
CNI RJ Moisturizer	https://pgmall.my/p/2740/6267	Selangor	Unknown	17.6
CNI RJ Hair Cream	https://pgmall.my/p/2740/6266	Selangor	Unknown	13.4
CNI RJ Shower Cream 300ml	https://pgmall.my/p/2740/6258	Selangor	Unknown	17.2
CNI Siang-Siang (100g) - Body Talc Absorbs Perspiration	https://pgmall.my/p/2740/6256	Selangor	Unknown	11.1
CNI RJ Intimate Wash	https://pgmall.my/p/2740/6251	Selangor	Unknown	18.8
ALHA ALFA ROYAL PROPOLIS LUMINOUS SILK FOUNDATIC	https://pgmall.my/p/2740/1287	Selangor	Penjagaan mulut	59.9
ALHA ALFA FLAWMINOUS CUSHION FOUNDATION	https://pgmall.my/p/2740/0510	Selangor	Alat solek	79.9
Xiaomi Smart Scale S200 | High-precision sensor | 180 (	https://pgmall.my/p/2740/0500	Selangor	Bekalan Perubatan	59
[BeautyVault] READY STOCK | RHODE - Pocket Blush	https://pgmall.my/p/2230/3788	Selangor	Alat solek	196
Dettol Shower Gel Refill Pouch 800ml/ 850ml / Dettol Ona	https://pgmall.my/p/Y714/0413	Selangor	Mandian	10.9
Zen Basil Seeds | edible basil seeds usda organic, kosh	https://pgmall.my/p/2739/4542	New Jersey	Makanan Tambahan	259
[BeautyVault] READY STOCK | ARIANA GRANDE FRAGRAI	https://pgmall.my/p/I226/0608	Selangor	Wangian	350
Morilins Pain Relief Patch 1bag 5pcs	https://pgmall.my/p/2739/4336	Johor	Bekalan Perubatan	13.35
SKINTIFIC 3x Acid Intensive Acne Spot Gel (15ml)	https://pgmall.my/p/2739/4075	Selangor	Penjagaan kulit	59
HAUS MAGICKISS GLITTER LIPSTICK 4G	https://pgmall.my/p/2739/4071	Selangor	Alat solek	27
HAUS POPSY SUPERSTAY LIPMATTE 3ML	https://pgmall.my/p/2739/4065	Selangor	Penjagaan kulit	27

Figure 4.3.1 Snapshot of Cleaned Data in "Item_list_cleaned.csv"

5.0 OPTIMIZATION TECHNIQUES

This section describes three techniques to clean and process data, each with different performance results. The first method uses standard Pandas. The other two optimization techniques applied on data processing are Dask - *parallel processing, improving efficiency on larger datasets*, and Polars - *fastest due to its optimized and vectorized operations*.

Table 5.0.1 Methods and description of optimization

Method	Descriptions
Pandas	• Uses standard Pandas operations • Processes data in-memory as a single chunk • Applies functions row-by-row using .apply() • Shows typical performance characteristics of unoptimized Pandas code
Dask	• Leverages Dask's parallel processing capabilities • Uses lazy evaluation with explicit computation triggers • Processes data in partitions for memory efficiency • Demonstrates distributed computing benefits
Polar	• Utilizes Polars' Rust-based, columnar data processing • Employs vectorized operations and expression-based transformations • Shows native string operations and regex optimizations • Demonstrates high-performance data frame operations

Code Overview

Pandas

```python
import pandas as pd
```

```python
# Price extraction function
def extract_price(price_str):
    matches = re.findall(r"\d+\.?\d*", str(price_str))
    return float(matches[0]) if matches else None

# Apply transformations
df['cleaned_price'] = df['price'].apply(extract_price).round(2)
df['product_name'] = df['product_name'].apply(lambda x: re.sub(r"[^\x00-\x7F]", "", str(x)))

# Finalize
df = df.rename(columns={
    'product_name': 'Product Name',
    'cleaned_price': 'Price',
    'location': 'Location',
    'link': 'Link',
    'product_type': 'Product Type'
}).drop(columns=['price'])
```

Dask

```python
import dask.dataframe as dd
```

```python
# Data cleaning pipeline
df = df.drop_duplicates()
df['location'] = df['location'].fillna('Unknown')
df['product_type'] = df['product_type'].fillna('Unknown')
df['cleaned_price'] = df['price'].map(extract_price, meta=('price', 'float'))
df['product_name'] = df['product_name'].map(clean_name, meta=('name', 'str'))

# Execute computation
result = df.compute()
result['Price'] = result['cleaned_price'].round(2)

# Finalize
result = result.rename(columns={
    'product_name': 'Product Name',
    'cleaned_price': 'Price',
    'location': 'Location',
    'link': 'Link',
    'product_type': 'Product Type'
}).drop(columns=['price'])
```

Polar

```python
import polars as pl

# Data cleaning
df = df.unique()  # Drop duplicates
df = df.with_columns([
    pl.col("location").fill_null("Unknown"),
    pl.col("product_type").fill_null("Unknown"),
    pl.col("price").str.extract(r"(\d+\.?\d*)").cast(pl.Float64).round(2).alias("Price"),
    pl.col("product_name").str.replace_all(r"[^\x00-\x7F]", "")
])

# Reorganize columns
df = df.select([
    pl.col("product_name").alias("Product Name"),
    "Price",
    pl.col("location").alias("Location"),
    pl.col("link").alias("Link"),
    pl.col("product_type").alias("Product Type")
])
```

6.0 PERFORMANCE EVALUATION

This section presents the performance comparison of the web scraping process and data processing before and after optimization. The evaluation metrics include total processing time, CPU usage, memory usage and throughput. The results were obtained by executing Python scripts that implemented the web scraping and data processing workflows, with performance data collected using built-in libraries within the Python environment.

Table 6.0.1 Performance Evaluation of Web Scraping

Metric	Without optimization	With optimization using multithreading
Total data scraped	200009 rows	200010 rows
Total time taken	16141.85 seconds (4 hours 28 minutes 58 seconds)	15517.04 seconds (4 hours 18 minutes 36 seconds)
Start CPU	2.5%	2.0%
End CPU	2.5%	3.5%
Start memory	166.88MB	167.01 MB
End memory	197.24 MB	274.76 MB
Used memory	30.36 MB	107.75 MB
Throughput	12.39 rec/s	12.89 rec/s

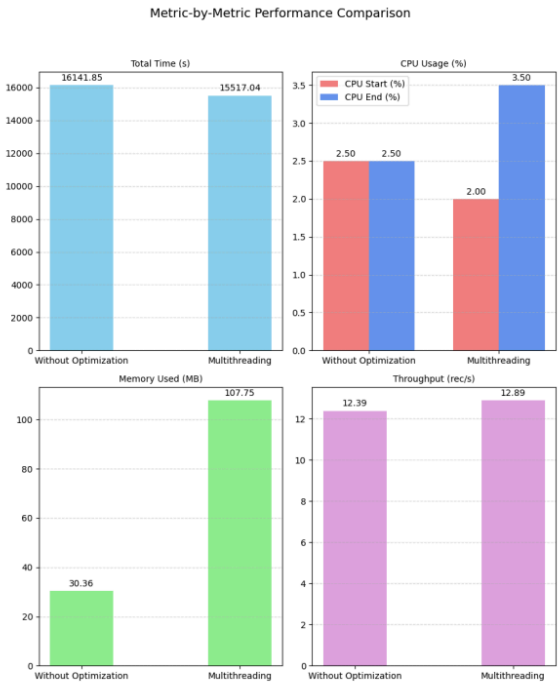

Figure 6.0.1 Performance Evaluation of Web Scraping Process Before and After Optimization

In the performance evaluation, the number of data obtained through web scraping with and without optimization were 200009 and 200010 respectively, which were nearly identical. From Table 6.0.1 and Figure 6.0.1, the total time taken for the web scraping with optimization is slightly faster than without optimization with nearly 4% improvement.

In terms of CPU usage, web scraping before optimization remained stable at 2.50% while the optimized version showed an increase from 2.0% to 3.5%. This indicates that multithreading utillizes more CPU resources to perform tasks concurrently. Besides, optimization also requires more memory usage, as shown in the evaluation. 107.75 MB memory was used during the optimized web scraping process while only 30.36 MB was used for basic web scraping. This suggests that multithreading introduces higher memory overhead due to concurrent execution.

The fourth bar chart illustrates the number of records processed per second, also known as throughput. The performance shows approximately a 4 % improvement in processing speed with multithreading.

Table 6.0.2 Performance Evaluation of Data Cleaning

Techniques	Metric	Run 1	Run 2	Run 3	Average
Pandas	Total processing time	4.03 seconds	2.89 seconds	2.80 seconds	3.24 seconds
	Memory usage	269.84 MB	243.20 MB	218.34MB	243.79MB
	CPU usage	0.0%	0.0%	0.0%	0.0%
	Throughput	51029.69 rec/s	71183.34 rec/s	51488.39 rec/s	57900.47 rec/s
Dask	Total processing time	6.70 seconds	6.86 seconds	6.38 seconds	6.65 seconds
	Memory usage	552.43 MB	447.21 MB	395.86 MB	465.17 MB
	CPU usage	1.0%	1.0%	0.0%	0.67 %
	Throughput	30719.88 rec/s	29994.33 rec/s	18149.05 rec/s	26287.75 rec/s
Polar	Total processing time	0.58 seconds	0.35 seconds	0.59 seconds	0.51 seconds
	Memory usage	319.55 MB	203.18 MB	240.30 MB	254.34 MB
	CPU usage	1.0%	1.0%	1.0%	1.0%
	Throughput	356363.62 rec/s	200205.37 rec/s	224245.89 rec/s	260271.63 rec/s

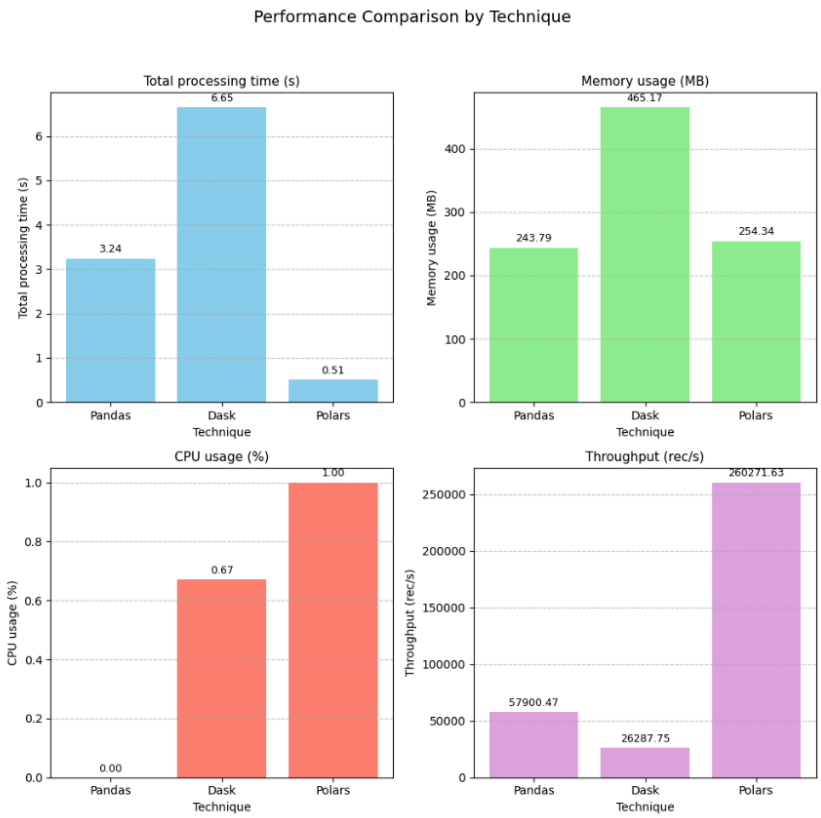

Figure 6.0.2 Performance Evaluation of Data Processing Before and After Optimization

Three different libraries – Pandas, Dask and Polars were used for performance evaluation in data cleaning. Among these three approaches, Polar was the fastest with an outstanding average result of 0.51 seconds while Dask took the longest average time at 6.65 seconds to complete the execution. Pandas performed better than Dask but still much slower compared to Polars. This illustrates that the advantage of Polars with its parallel processing offers faster processing speed.

Pandas consumed the least average amount of memory (243.79 MB), followed by Polars and then Dask. This shows that Pandas is memory efficient for in-memory operations while Dask needs higher memory usage to operate multiple partitions in parallel.

In terms of CPU usage, Pandas showed minimal CPU utilization due to its single-threaded limitation. Meanwhile, Dask and Polars with parallel processing utilized more CPUs. Besides, Polars had the highest throughput; Pandas had an average throughput of 57900.47 records per second; Dask had the lowest throughput.

Overall, Polars performed the best in this evaluation. Dask might not be the best approach for the data cleaning of this dataset. The size of the dataset around 200K records is considered small. Thus, it is as expected that Polars is the fastest as it is featured with a multi-threaded query engine and columnar processing that enables high-performance processing (Polars 2025). The concept of Dask is to build a task graph to schedule task execution on a single machine (Anaconda, Inc. , 2018). This feature is beneficial for Big Data or large-scale data processing on a single machine(Singh, 2024). However, it is not suitable for small datasets like the one used in this project because extra time and resources are needed for Dask to build a task graph. Therefore, the processing time of Dask is the longest and takes the highest memory usage among the three techniques. Both Polar and Dask are designed to fully utilize the available CPU cores; thus, it matches the trend observed in the evaluation.

7.0 CHALLENGES & LIMITATIONS

This section covers the key difficulties faced during the web scraping project, including technical barriers, data availability issues, and hardware limitations. Not all websites could be scrapped due to anti-bot protections and many Malaysian sites had smaller datasets than expected.

Table 7.0.1 Challenges of project

Challenges	Descriptions
Website Restrictions	Some websites block scraping via CAPTCHAs, rate limiting and dynamic content
Limited Data Availability	Many Malaysian websites have small datasets and some sites paginate data poorly, making large-scale scraping difficult.
Inconsistent Data Structure	Websites change layout, breaking scrapers and some data is hidden behind login walls

Table 7.0.2 Limitations of project

Limitations	Descriptions
Hardware Dependencies	Scraping speed varies by CPU/RAM and low-end devices struggle with large-scale scraping
Slow Scraping Process	Polar/Dask clean data fast, but scraping itself bottlenecked by network latency and rate-limiting delays
Maintenance Overload	Scarpers need constant updates if websites change HTML structure and proxy/IP rotation may be needed to avoid bans

8.0 CONCLUSION

This project demonstrated the development of high-performance data processing for large-scale web crawling from PG Mall's "Health & Beauty" category, which successfully gathered over 200,000 cleaned records. This exceeds the minimum requirement for 100,000 records. The system applied multithreaded scraping to compare with the basic scraping in order to reach a more efficient performance. It can be concluded that multithreading is a better choice when prioritising the performance as it offers faster data collection and higher throughput. The downsides are the increased CPU and memory usage. However, the trade-off is acceptable in this case with a larger volume of data.

In the data cleaning process, the three frameworks - Pandas, Dask and Polars are compared in terms of performance evaluation. It can be seen that Polars is the most efficient and scalable option for data cleaning in high-performance cases. It significantly outperformed Pandas and Dask in speed and throughput, making it ideal for large-scale pipelines. Pandas remains useful for simplicity and low-memory environments, while Dask is better suited for much larger or distributed records where partitioned processing can be better leveraged.

Although the final result from this project is successful, several areas present opportunities for improvement to tackle the challenges faced throughout the project development. First, to handle website restrictions like CAPTCHA and rate limiting, the scraper could be enhanced with proxy rotation and headless browsers to better mimic real users. For websites with limited or poorly paginated data, expanding the scope to include multiple categories or additional websites could help gather more useful data. To reduce scraping delays caused by network latency, the use of asynchronous requests could speed up the process further. Hardware limitations can be addressed by running the scraping process on a more powerful and stable cloud environment such as Google Cloud Compute Engine or AWS EC2. Lastly, to reduce the need for constant maintenance due to layout changes, the scraper can be improved with more adaptive or modular code and switched to using public APIs for more stable and structured access to data.

9.0 REFERENCES

Anaconda, Inc. (2018). *10 minutes to dask*. 10 Minutes to Dask - Dask documentation. https://docs.dask.org/en/stable/10-minutes-to-dask.html

Polars. (2025). *Polars*. Polars -. https://pola.rs/

Singh, A. (2024, October 21). Ultimate Guide to handle big datasets for machine learning using DASK (in python). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/08/dask-big-datasets-machine_learning-python/#h-3-introduction-to-dask

10.0 APPENDICES

Appendix A Code Snippets

Web Scraping

```
import requests
from bs4 import BeautifulSoup
import csv
from google.colab import files
import time
from datetime import datetime
import os
import psutil

# Fetch the webpage
url_template = "https://pgmall.my/category?path=1&page={}"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36"
}

performance_log_file = "performance_log.csv"
max_data_limit = 200000
version_label = "Without Optimization"

# Performance tracking setup
process = psutil.Process(os.getpid())
start_cpu = psutil.cpu_percent(interval=1)
start_mem = process.memory_info().rss / (1024 * 1024)  # MB
```

```
def scrape_page(page_number):
    """Scrape a single page with retry logic"""
    url = url_template.format(page_number)
    retries = 3

    for attempt in range(retries):
      response = requests.get(url, headers=headers)
      if response.status_code == 429:
          retry_after = int(response.headers.get('Retry-After', 5))
          print(f"Rate limited on page {page_number}, retrying in {retry_after} seconds... (Attempt {attempt+1}/{retries})")
          time.sleep(retry_after)
          continue
      elif response.status_code != 200:
          print(f"Failed to fetch page {page_number}. Status code: {response.status_code}")
          return []

      soup = BeautifulSoup(response.content, 'html.parser')
      listings = soup.find_all('div', class_='category_product_col_new p-div')

      data = []
      for listing in listings:
          # Extract the product link
          link_tag = listing.find('a', href=True)
          link = link_tag['href'] if link_tag else None

          # Extract the product name
          name_tag = listing.find('p', class_='p-name text-left text-darkgrey')
          product_name = name_tag.text.strip() if name_tag else None

          # Extract price
          price_tag = listing.find('span', class_='p-price-red p-overflow')
          price = price_tag.text.strip() if price_tag else None
```

```
            # Extract location
            location_tag = listing.find('div', class_='text-left color-grey')
            location = location_tag.text.strip() if location_tag else None

            # Append the extracted data to the list
            data.append({
                "product_name": product_name,
                "link": link,
                "price": price,
                "location": location,
            })

        return data

def save_to_csv(data, csv_file="Item_list.csv"):
    file_exists = os.path.isfile(csv_file)
    with open(csv_file, mode='a', newline='', encoding='utf-8') as f:
        writer = csv.DictWriter(f, fieldnames=data[0].keys())
        if not file_exists:
            writer.writeheader()
        writer.writerows(data)
```

Data Cleaning and Processing

```
import pandas as pd
import re
import time
import psutil
import os

# Load dataset
df = pd.read_csv("updated_item_list.csv")

# Log initial shape
initial_shape = df.shape
```

```python
# Clean price column and extract lowest numeric value
def extract_lowest_price(price_str):
    if pd.isna(price_str):
        return None
    matches = re.findall(r"\d+(?:\.\d+)?", price_str)
    if matches:
        return float(matches[0])
    return None
```

```python
# Start performance timer and process monitor
start_time = time.time()
process = psutil.Process(os.getpid())
# Drop duplicate rows
df = df.drop_duplicates()

# Fill in null
df['location'].fillna('Unknown', inplace=True)
df['product_type'].fillna('Unknown', inplace=True)

# Standardize price format
df['cleaned_price'] = df['price'].apply(extract_lowest_price)
df['cleaned_price'] = df['cleaned_price'].round(2)

df.drop(columns=['price'], inplace=True)

# Clean unreadable characters from product_name
df['product_name'] = df['product_name'].apply(lambda x: re.sub(r"[^\x00-\x7F]+", '', str(x)))
```

```python
df.rename(columns={
    'product_name': 'Product Name',
    'cleaned_price': 'Price',
    'location': 'Location',
    'link': 'Link',
    'product_type': 'Product Type'
}, inplace=True)

# Save cleaned data
df.to_csv("Item_list_cleaned.csv", index=False, float_format='%.2f')

# Log performance metrics
end_time = time.time()
elapsed_time = end_time - start_time
cpu_percent = process.cpu_percent(interval=1)
memory_usage_mb = process.memory_info().rss / 1024 ** 2
throughput = df.shape[0] / elapsed_time

print("\n--- Performance Metrics ---")
print(f"Total processing time: {elapsed_time:.2f} seconds")
print(f"CPU usage: {cpu_percent}%")
print(f"Memory usage: {memory_usage_mb:.2f} MB")
print(f"Throughput: {throughput:.2f} records/second")
print(f"Rows before: {initial_shape[0]}, Rows after: {df.shape[0]}")

from google.colab import files
files.download("Item_list_cleaned.csv")
```

```python
def log_performance(version, total_data_saved, elapsed_time, start_cpu, end_cpu, start_mem, end_mem):
    throughput = total_data_saved / elapsed_time if elapsed_time > 0 else 0
    with open(performance_log_file, "a", newline="") as f:
        writer = csv.writer(f)
        if f.tell() == 0:
            writer.writerow(["version", "total_records", "total_time", "cpu_start", "cpu_end", "mem_start", "mem_end", "throughput"])
        writer.writerow([
            version,
            total_data_saved,
            elapsed_time,
            start_cpu,
            end_cpu,
            start_mem,
            end_mem,
            throughput
        ])
```

```python
def main():
    page_number = 1
    total_data_saved = 0
    request_delay = 0
    start_time = time.time()

    while True:
        print(f"Scraping page {page_number}...")
        property_data = scrape_page(page_number)

        try:
          if not property_data:
              print(f"No data found on page {page_number}. Exiting...")
              break

          save_to_csv(property_data)
          total_data_saved += len(property_data)
          print(f"Page {page_number} scraped and saved ({len(property_data)} items). Total: {total_data_saved}")
          if total_data_saved >= max_data_limit:
              print(f"Reached the max data limit of {max_data_limit}. Exiting...")
              break

          page_number += 1
          time.sleep(request_delay)
        except Exception as e:
          print(f"Error scraping page {page_number}: {e}")
          break

    end_time = time.time()
    elapsed_time = end_time - start_time
```

```
    # End performance stats
    end_cpu = psutil.cpu_percent(interval=1)
    end_mem = process.memory_info().rss / (1024 * 1024)

    # Print results
    print(f"\nTotal records scraped: {total_data_saved}")
    print(f"Total time taken: {elapsed_time:.2f} seconds")
    print(f"Start memory: {start_mem:.2f} MB, End memory: {end_mem:.2f} MB")
    print(f"Start CPU: {start_cpu}%, End CPU: {end_cpu}%")
    print(f"Records per second: {total_data_saved / elapsed_time:.2f} rec/sec")

    # Log to performance file
    log_performance(version_label, total_data_saved, elapsed_time, start_cpu, end_cpu, start_mem, end_mem)

        # Final cleanup and download (only in Google Colab)
    if os.path.isfile(csv_file):
        if 'google.colab' in str(get_ipython()):
            files.download(csv_file)
            if os.path.isfile(performance_log_file):
              files.download(performance_log_file)
        else:
            print(f"CSV file saved locally: {csv_file}")
    else:
        print("No data was scraped. CSV file was not created.")

if __name__ == "__main__":
    main()
```

Chart of Performance Evaluation

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Labels
metrics = ['Total Time (s)', 'CPU Start (%)', 'CPU End (%)', 'Memory Used (MB)', 'Throughput (rec/s)']
without_opt = [16141.85, 2.5, 2.5, 197.24 - 166.88, 12.39]
multithreading = [15517.04, 2.0, 3.5, 274.76 - 167.01, 12.89]
versions = ['Without Optimization', 'Multithreading']

# Plotting each metric in a separate subplot (2 rows, 2 columns for 4 plots)
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(9, 11), sharey=False)
axes = axes.flatten()  # Flatten 2D axes array for easier indexing

# Index reference for CPU subplot (combined CPU Start and End)
cpu_index = 1  # second in original list, now becomes 1st in new layout

colors = ['skyblue', 'salmon', 'lightgreen', 'plum']

for i, ax in enumerate(axes):
    # Adjust the metric index because CPU Start and End are combined
    metric_index = i if i < cpu_index else i + 1

    if i == cpu_index:
        # Combined CPU Start and End
        x = range(len(versions))
        bar_width = 0.35

        start_values = [without_opt[1], multithreading[1]]
        end_values = [without_opt[2], multithreading[2]]

        bars1 = ax.bar([p - bar_width/2 for p in x], start_values, width=bar_width, label='CPU Start (%)', color='lightcoral')
        bars2 = ax.bar([p + bar_width/2 for p in x], end_values, width=bar_width, label='CPU End (%)', color='cornflowerblue')
```

```
        ax.set_xticks(x)
        ax.set_xticklabels(versions)
        ax.set_title('CPU Usage (%)', fontsize=10)
        ax.grid(axis='y', linestyle='--', alpha=0.5)
        ax.legend()

        for bars in [bars1, bars2]:
            for bar in bars:
                height = bar.get_height()
                ax.annotate(f'{height:.2f}',
                            xy=(bar.get_x() + bar.get_width()/2, height),
                            xytext=(0, 3),
                            textcoords='offset points',
                            ha='center', va='bottom')
    else:
        # Regular metric bar plot
        values = [without_opt[metric_index], multithreading[metric_index]]
        bars = ax.bar(versions, values, color=colors[i], width=0.4)
        ax.set_title(metrics[metric_index], fontsize=10)
        ax.grid(axis='y', linestyle='--', alpha=0.5)

        for bar in bars:
            height = bar.get_height()
            ax.annotate(f'{height:.2f}',
                        xy=(bar.get_x() + bar.get_width()/2, height),
                        xytext=(0, 3),
                        textcoords='offset points',
                        ha='center', va='bottom')

plt.suptitle('Metric-by-Metric Performance Comparison', fontsize=14)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

```
data = {
    'Technique': ['Pandas', 'Dask', 'Polars'],
    'Total processing time (s)': [
        [4.03, 2.89, 2.80],
        [6.70, 6.86, 6.38],
        [0.58, 0.35, 0.59]
    ],
    'Memory usage (MB)': [
        [269.84, 243.20, 218.34],
        [552.43, 447.21, 395.86],
        [319.55, 203.18, 240.30]
    ],
    'CPU usage (%)': [
        [0.0, 0.0, 0.0],
        [1.0, 1.0, 0.0],
        [1.0, 1.0, 1.0]
    ],
    'Throughput (rec/s)': [
        [51029.69, 71183.34, 51488.39],
        [30719.88, 29994.33, 18149.05],
        [356363.62, 200205.37, 224245.89]
    ]
}

# Create a DataFrame from the average of each metric
avg_data = {
    'Technique': data['Technique'],
    'Total processing time (s)': [round(sum(val) / len(val), 2) for val in data['Total processing time (s)']],
    'Memory usage (MB)': [round(sum(val) / len(val), 2) for val in data['Memory usage (MB)']],
    'CPU usage (%)': [round(sum(val) / len(val), 2) for val in data['CPU usage (%)']],
    'Throughput (rec/s)': [round(sum(val) / len(val), 2) for val in data['Throughput (rec/s)']]
}

# Create a DataFrame
df = pd.DataFrame(avg_data)

print(df)
```

```
# Set figure size
plt.figure(figsize=(14, 10))

# Plot each metric in a subplot
metrics = ['Total processing time (s)', 'Memory usage (MB)', 'CPU usage (%)', 'Throughput (rec/s)']
colors = ['skyblue', 'lightgreen', 'salmon', 'plum']

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
axes = axes.flatten()

for i, (ax, metric) in enumerate(zip(axes, metrics)):
    bars = ax.bar(df['Technique'], df[metric], color=colors[i])
    ax.set_title(metric, fontsize=11)
    ax.set_ylabel(metric)
    ax.set_xlabel('Technique')
    ax.grid(axis='y', linestyle='--', alpha=0.7)


    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords='offset points',
                    ha='center', va='bottom', fontsize=9)

plt.suptitle('Performance Comparison by Technique', fontsize=14)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Appendix B Screenshots of Output

Output of Performance Evaluation - Web Scraping

```
Page 3994 scraped and saved (50 items). Total: 199610
Scraping page 3995...
Page 3995 scraped and saved (50 items). Total: 199660
Scraping page 3996...
Page 3996 scraped and saved (50 items). Total: 199710
Scraping page 3997...
Page 3997 scraped and saved (50 items). Total: 199760
Scraping page 3998...
Page 3998 scraped and saved (50 items). Total: 199810
Scraping page 3999...
Page 3999 scraped and saved (50 items). Total: 199860
Scraping page 4000...
Page 4000 scraped and saved (50 items). Total: 199910
Scraping page 4001...
Page 4001 scraped and saved (50 items). Total: 199960
Scraping page 4002...
Page 4002 scraped and saved (50 items). Total: 200010
Reached the maximum data limit of 200000. Exiting...

Total records scraped: 200010
Total time taken: 15517.04 seconds
Start memory: 167.01 MB, End memory: 274.76 MB
Start CPU: 2.0%, End CPU: 3.5%
Records per second: 12.89 rec/sec
```

```
Page 3992 scraped and saved (50 items). Total: 199509
Scraping page 3993...
Page 3993 scraped and saved (50 items). Total: 199559
Scraping page 3994...
Page 3994 scraped and saved (50 items). Total: 199609
Scraping page 3995...
Page 3995 scraped and saved (50 items). Total: 199659
Scraping page 3996...
Page 3996 scraped and saved (50 items). Total: 199709
Scraping page 3997...
Page 3997 scraped and saved (50 items). Total: 199759
Scraping page 3998...
Page 3998 scraped and saved (50 items). Total: 199809
Scraping page 3999...
Page 3999 scraped and saved (50 items). Total: 199859
Scraping page 4000...
Page 4000 scraped and saved (50 items). Total: 199909
Scraping page 4001...
Page 4001 scraped and saved (50 items). Total: 199959
Scraping page 4002...
Page 4002 scraped and saved (50 items). Total: 200009
Reached the max data limit of 200000. Exiting...

Total records scraped: 200009
Total time taken: 16141.85 seconds
Start memory: 166.88 MB, End memory: 197.24 MB
Start CPU: 2.5%, End CPU: 2.5%
Records per second: 12.39 rec/sec
```

Output of Performance Evaluation - Data Cleaning

Run 1

```
--- Performance Metrics ---
Total processing time: 4.03 seconds
CPU usage: 0.0%
Memory usage: 269.84 MB
Throughput: 51029.69 records/second
```

```
--- Dask Optimized Performance ---
Total processing time: 6.70 seconds
CPU usage: 1.0%
Memory usage: 552.43 MB
Throughput: 30719.88 records/second
```

```
--- Optimized Performance Metrics (Polars) ---
Total processing time: 0.58 seconds
CPU usage: 1.0%
Memory usage: 319.55 MB
Throughput: 356363.62 records/second
```

Run 2

```
--- Performance Metrics ---
Total processing time: 2.89 seconds
CPU usage: 0.0%
Memory usage: 243.20 MB
Throughput: 71183.34 records/second
```

```
--- Dask Optimized Performance ---
Total processing time: 6.86 seconds
CPU usage: 1.0%
Memory usage: 447.21 MB
Throughput: 29994.33 records/second
```

```
--- Optimized Performance Metrics (Polars) ---
Total processing time: 0.35 seconds
CPU usage: 1.0%
Memory usage: 203.18 MB
Throughput: 200205.37 records/second
```

Run 3

```
--- Performance Metrics ---
Total processing time: 2.80 seconds
CPU usage: 0.0%
Memory usage: 218.34 MB
Throughput: 51488.39 records/second
```

```
--- Dask Optimized Performance ---
Total processing time: 6.38 seconds
CPU usage: 0.0%
Memory usage: 395.86 MB
Throughput: 18149.05 records/second
```

```
--- Optimized Performance Metrics (Polars) -
Total processing time: 0.59 seconds
CPU usage: 1.0%
Memory usage: 240.30 MB
Throughput: 224245.89 records/second
```

Appendix C Links to Full Code & Dataset

Link to Full Code Folder:
https://drive.google.com/drive/folders/1LxHqnUEUspyp9mxUi2mLtvIscrrFHdZG?usp=sharing

Link to Datasets: https://github.com/drshahizan/HPDP/tree/main/2425/project/p1/GroupD/data

HPDP - PROJECT 1 REPORT.pdf

ORIGINALITY REPORT

3%	1%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Dzikiti, Lister Munodawafa. "Teachers' Pedagogical Content Knowledge and Learners' Performance in Electromagnetic Induction Following a Generic Intervention", University of Pretoria (South Africa), 2024 Publication	1%
2	www.coursehero.com Internet Source	1%
3	Submitted to Glyndwr University Student Paper	<1%
4	Submitted to Universiti Teknologi Malaysia Student Paper	<1%
5	easyschool.ru Internet Source	<1%
6	hdl.handle.net Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		