SECP3133-02

HIGH-PERFORMANCE DATA PROCESSING

PROJECT 1: OPTIMIZING HIGH-PERFORMANCE DATA PROCESSING FOR LARGE-SCALE WEB CRAWLERS

Prepared By:

NURUL ERINA BINTI ZAINUDDIN A22EC0254
ONG YI YAN A22EC0101
TANG YAN QING A22EC0109
WONG QIAO YING A22EC0118

Lecturer Name:

DR. ARYATI BINTI BAKRI

Submission Date:

16 May 2025

1. INTRODUCTION

1.1. Background Of The Project

The ability to rapidly gather and process vast amounts of web data has become crucial in the big data era, particularly in domains like analytics and data science. Web crawling is a popular method for collecting data from websites, although it frequently has technical drawbacks like poor performance and problems with data quality.

This project's main goal is to develop a web crawler capable of extracting a sizable dataset specifically from eBay, leveraging High-Performance Computing (HPC) techniques such as threading, asyncio, and multiprocessing. The project also aims to compare the performance of these techniques using four different libraries: Polars, Pandas, PySpark, and Modin. Through this process, practical experience in web data extraction, data cleansing, and performance optimization will be gained.

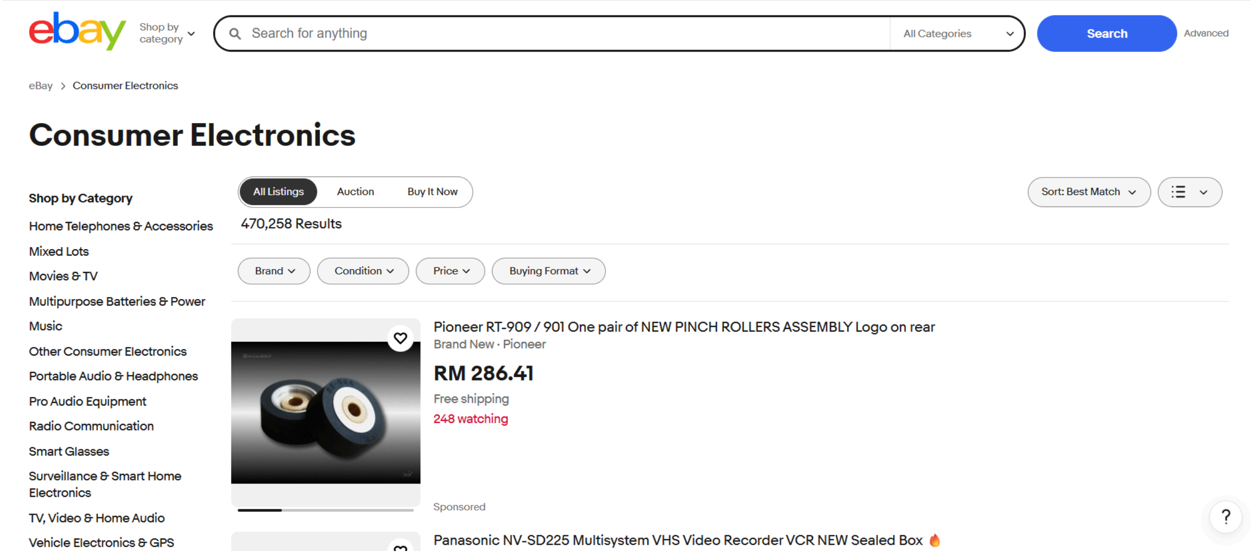
1.2. Objectives

1. To develop and implement a web crawler using four different libraries: Playwright, Requests+BeautifulSoup, Scrapy, and Selenium to extract at least 100,000 structured property-related records from eBay.

2. To process and clean the data using high-performance computing techniques using different data processing libraries: Polars, Pandas, PySpark, and Modin while applying HPC techniques such as threading, asyncio, and multiprocessing.

3. To perform a performance comparison of the different scraping and processing approaches in terms of total time, memory usage, CPU efficiency, and throughput.

1.3. Target Website And Data To Be Extracted



Consumer Electronics

Shop by Category

Home Telephones & Accessories
Mixed Lots
Movies & TV
Multipurpose Batteries & Power
Music
Other Consumer Electronics
Portable Audio & Headphones
Pro Audio Equipment
Radio Communication
Smart Glasses
Surveillance & Smart Home Electronics
TV, Video & Home Audio
Vehicle Electronics & GPS

470,258 Results

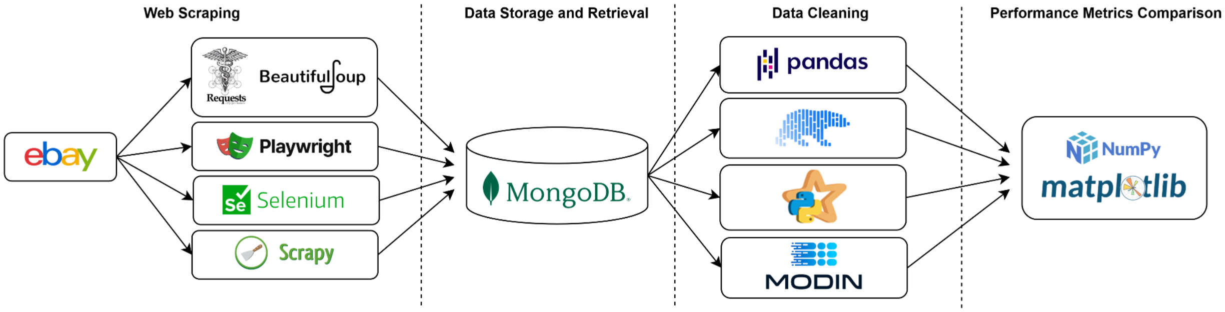
Pioneer RT-909 / 901 One pair of NEW PINCH ROLLERS ASSEMBLY Logo on rear
Brand New · Pioneer
RM 286.41
Free shipping
248 watching
Sponsored

Panasonic NV-SD225 Multisystem VHS Video Recorder VCR NEW Sealed Box

eBay.com.my is the Malaysian portal of the global e-commerce platform eBay, where users can buy and sell a wide range of items, from electronics to fashion and collectibles. We specifically extract product listings from eBay. From these listings, we extract the **Title**, which is the key identifier of the product; the **Price**, showing the listed selling price; the **Shipping Fee**, indicating the delivery cost or free shipping offers; the **Link**, which provides the direct URL to the product page; as well as the **Category**, identifying the product type or classification; the **Brand**, indicating the manufacturer; and the **Condition**, describing whether the item is new or used.

2. SYSTEM DESIGN & ARCHITECTURE

2.1. Description of Architecture (include diagram)



2.2. Tools and Frameworks Used

- Integrated Development Environment (IDE) for code development and testing in the cloud:
    - Google Colab
- Core language for crawling, data processing, and analysis
    - Python
- Python libraries for web scraping
    - Requests
    - BeautifulSoup
    - Selenium
    - Playwright
    - Scrapy
- Python libraries for data cleaning
    - Pandas
    - Polars
    - PySpark
    - Mobin
- Database
    - MongoDB Atlas
- Create architecture design
    - Draw.io
- Reporting & documentation
    - Google Docs

2.3. Roles of Team Members

Process	**Member Name**	**Outcome/ Description**
Website identification	Everyone	eBay Malaysia
Identify libraries for web scraping	Nurul Erina Binti Zainuddin	Selenium
	Ong Yi Yan	Scrapy
	Tang Yan Qing	Playwright
	Wong Qiao Ying	Requests, BeautifulSoup
Web Scraping	Everyone	Each member scrape at least 25,000 rows of data
Database Setup	Wong Qiao Ying	MongoDB Atlas
Data Cleaning	Nurul Erina Binti	Modin

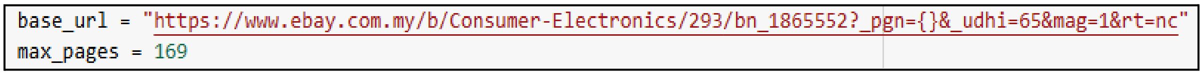
	Zainuddin	
	Ong Yi Yan	PySpark
	Tang Yan Qing	Polars
	Wong Qiao Ying	Pandas
Performance Metrics Comparison	Ong Yi Yan	Matplotlib, Numpy
Reporting and Documentation	Nurul Erina Binti Zainuddin	● Crawling Method and Number of Records ● Conclusion & Future Work ● Appendices
	Ong Yi Yan	● Performance Evaluation ● Data Processing
	Tang Yan Qing	● Data Collection (Ethical Considerations) ● Optimization Techniques ● Challenges and Limitations
	Wong Qiao Ying	● Description of Architecture ● Data Processing ● References

3. DATA COLLECTION

3.1. Crawling method (pagination, rate-limiting, async)

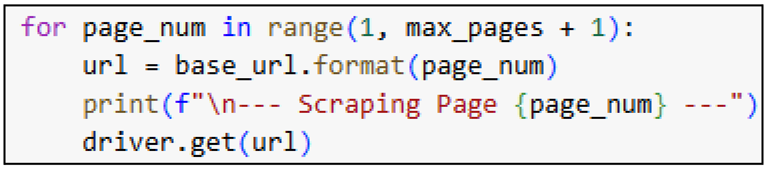
3.1.1 Pagination Handling

This script uses pagination by changing the page number in the URL to scrape eBay listings one page at a time. Since eBay only shows real data up to page 169, the script also uses a price filter (`_udhi=65`), (`_udhi=150 & udlo=65`) and (`udlo=150`) to limit results and stay within that range.



```
base_url = "https://www.ebay.com.my/b/Consumer-Electronics/293/bn_1865552?_pgn={}&_udhi=65&mag=1&rt=nc"
max_pages = 169
```

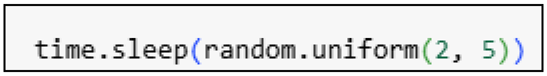
This loop visits each eBay page (up to 169) with a price filter set



```
for page_num in range(1, max_pages + 1):
    url = base_url.format(page_num)
    print(f"\n--- Scraping Page {page_num} ---")
    driver.get(url)
```

3.1.2 Rate-Limiting

The script uses rate limiting by adding random delays between page requests to avoid detection or blocking by eBay. This mimics human behavior and makes the scraping process appear more natural



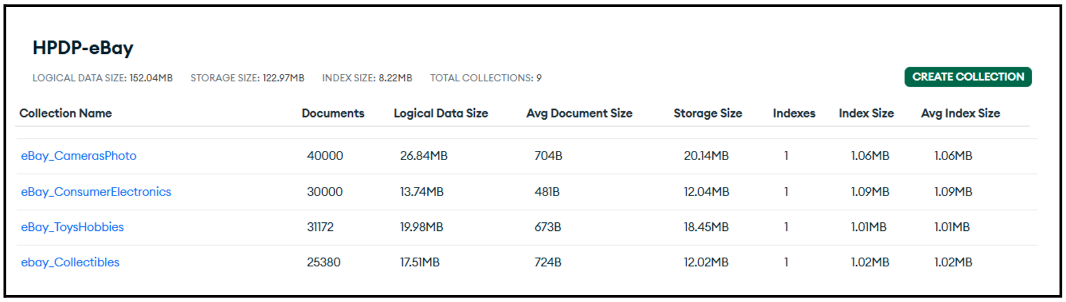
```
time.sleep(random.uniform(2, 5))
```

3.1.3 Asynchronous Support (Not used)

All the commands (driver.get(url) or find_element) are executed one after the other, and the program doesn't continue until the previous step is complete. There are no asynchronous methods involved, and everything is done one after another, one page and one listing at the time.

3.2. Number of records collected

Total of `126552` rows of data has been extracted and stored in mongodb.



HPDP-eBay

LOGICAL DATA SIZE: 152.04MB STORAGE SIZE: 122.97MB INDEX SIZE: 8.22MB TOTAL COLLECTIONS: 9

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
eBay_CamerasPhoto	40000	26.84MB	704B	20.14MB	1	1.06MB	1.06MB
eBay_ConsumerElectronics	30000	13.74MB	481B	12.04MB	1	1.09MB	1.09MB
eBay_ToysHobbies	31172	19.98MB	673B	18.45MB	1	1.01MB	1.01MB
ebay_Collectibles	25380	17.51MB	724B	12.02MB	1	1.02MB	1.02MB

3.3. Ethical considerations

While conducting data scraping on eBay, we were highly conscious of the ethical and legal boundaries. First of all, scraping was carried out solely for research and educational purposes only. We had no intention of using it for commercial purposes or data misuse. Second, we collected only publicly published product details like brand, title, category, price and shipping fee without touching any kind of sensitive or transactional user information. Third, to ensure minimal server load and mimic real user activity, we utilized throttling techniques, including randomized user agents, hold times and controlled request rates. Fourth, we also reviewed eBay's robots.txt file to guide responsible scraping activity and kept our tool usage below acceptable levels. Transparency and integrity were paramount throughout the duration of the project.

4. Data Processing

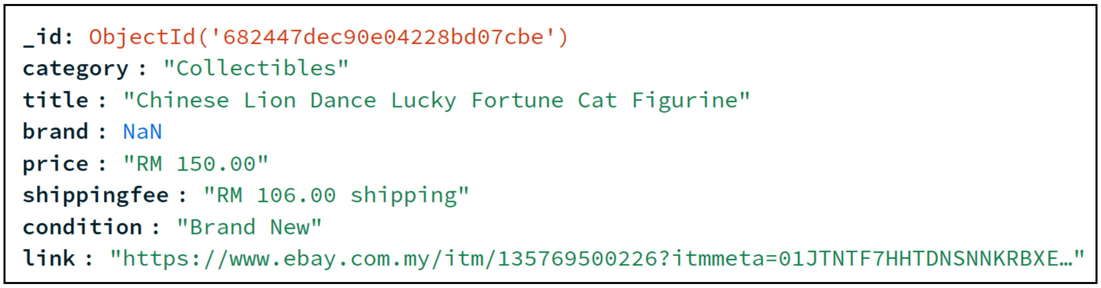
4.1. Cleaning methods

The following cleaning techniques were applied:

- Merge four distinct collections from a MongoDB database into a single, unified DataFrame.
- Drop the _id column from the dataset.
- Convert floating-point NaN (Not a Number) values across all columns to null for consistent missing data handling.
- Remove rows containing three or more null values (specifically, those with fewer than (total number of columns) - 2 non-null values).
- Fill remaining null values in specific columns with predefined defaults: 'Unknown' for 'brand', 'category', 'condition', and 'link'; 'Unknown' (corrected from 'Unknown') for 'title'; and 0.0 for 'shippingfee' and 'price'.
- Remove duplicate rows based on the combination of 'title', 'brand', 'price', 'shippingfee', and 'condition' columns.
- Trim leading and trailing whitespaces from all string-type columns.
- Normalize eBay URLs in the 'link' column to a canonical format by extracting the item ID.
- Clean the 'title' column by removing 'NEW LISTING' prefixes, sanitizing content (removing most symbols and non-alphanumeric characters), and performing a final whitespace trim.

4.2. Data structure (CSV/JSON/database)

After web scraping, each of the categories are stored as different collections in MongoDB under the database named HPDP_eBay,, namely eBay_CamerasPhoto, eBay_Collectibles, eBay_ToysHobbies, eBay_ConsumerElectronics. Data below shows the structure of the collection before cleaning. It consists of nine attributes.



```
_id: ObjectId('682447dec90e04228bd07cbe')
category : "Collectibles"
title : "Chinese Lion Dance Lucky Fortune Cat Figurine"
brand : NaN
price : "RM 150.00"
shippingfee : "RM 106.00 shipping"
condition : "Brand New"
link : "https://www.ebay.com.my/itm/135769500226?itmmeta=01JTNTF7HHTDNSNNKRBXE…"
```

After retrieving each collection, it was transformed into a Pandas DataFrame and then merged into a single DataFrame for data processing. Before the dataset was saved or exported, this structure allowed for flexible manipulation and cleaning.

4.3. Transformation and formatting

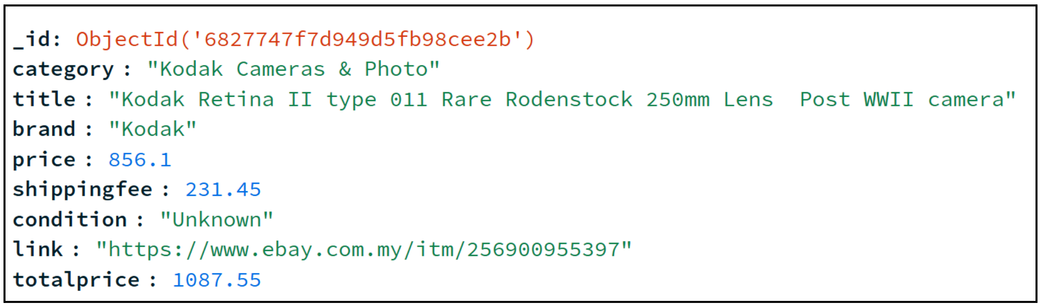
The following data processing steps were performed:

- Transform the 'shippingfee' column by extracting numeric values (e.g., converting 'RM 23.00 shipping' to 23.00) and standardizing 'free shipping' entries to 0.0, ensuring the column becomes a numeric type.

- Format the 'price' column by removing currency symbols or prefixes (e.g., converting 'rm23.00' to 23.00) and converting it to a numeric data type.

- Create a new 'totalprice' column by summing the numerically formatted 'price' and 'shippingfee' columns.

Figure below displays the data after cleaning.



```
_id: ObjectId('6827747f7d949d5fb98cee2b')
category : "Kodak Cameras & Photo"
title : "Kodak Retina II type 011 Rare Rodenstock 250mm Lens  Post WWII camera"
brand : "Kodak"
price : 856.1
shippingfee : 231.45
condition : "Unknown"
link : "https://www.ebay.com.my/itm/256900955397"
totalprice : 1087.55
```

5. Optimization Techniques

5.1. Methods used

5.1.1. PySpark

PySpark enhances large-scale big data processing by distributing tasks across multiple CPU cores or machines. This enables parallel execution and in-memory computation, significantly enhancing performance when working with big data. Operations such as deduplication, management of missing values and the addition of new columns are done lazily, which means they are only processed when needed. This enables Spark's Catalyst optimizer to develop optimal plans of execution. Additionally, complex operations with regular expressions such as replacements and extractions are performed directly on Spark DataFrames, reducing unnecessary intermediate computations and improving memory efficiency.

5.1.2. Modin

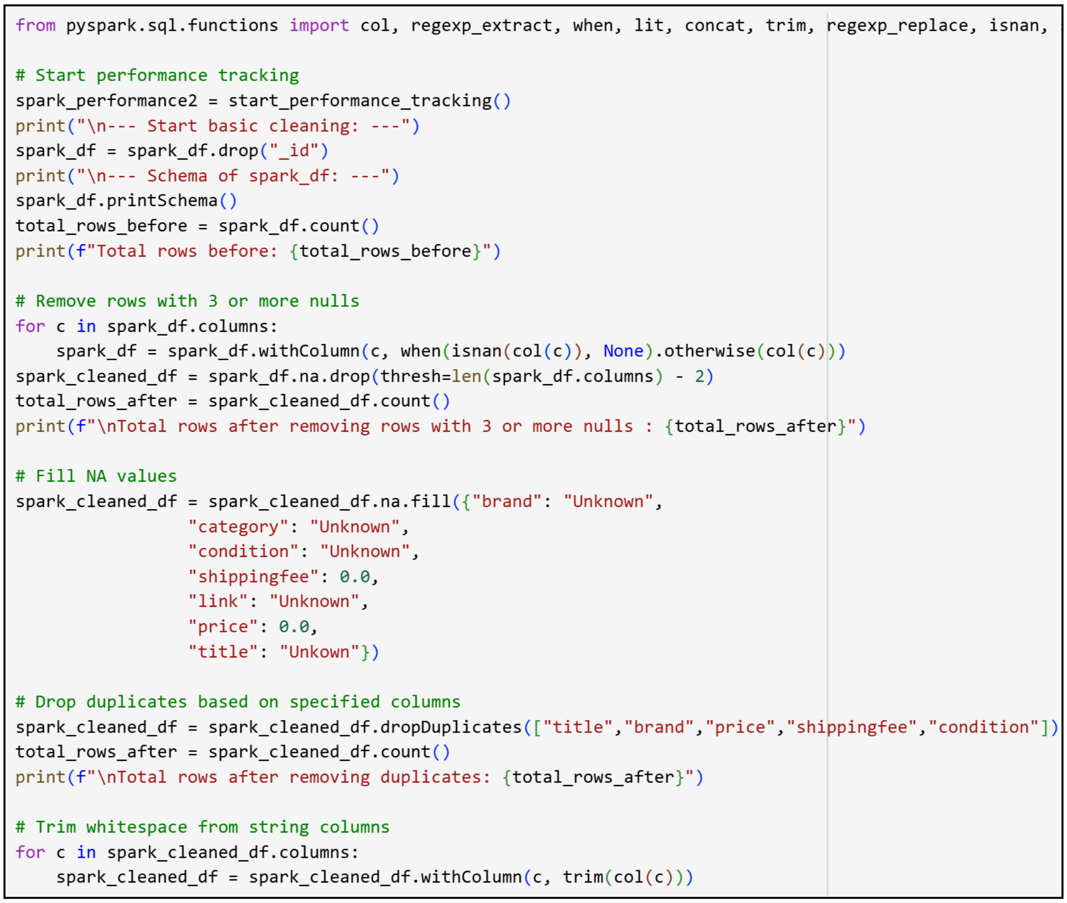
Modin improves the performance of pandas operations by distributing them across multiple CPU cores using multiprocessing frameworks like Ray or Dask. Pandas operations such as reading the data, combining DataFrames, dropping duplicates and substituting strings are automatically parallelized, which makes the execution faster with minimal changes to existing code. Modin preserves the familiar pandas interface while internally managing data partitioning and task scheduling. It also defers execution and optimizes memory usage by chunking operations, providing a scalable and efficient solution for large datasets in multi-core environments.

5.1.3. Polars

Polars utilizes multithreading and SIMD (Single Instruction, Multiple Data) to accelerate data processing on a single machine. Through its lazy evaluation API, multiple operations such as column transformations, filtering, and regex-based cleaning are combined into a single optimized execution plan and run in parallel. This minimizes overhead from intermediate computations. Polars supports streaming execution as well, so it can handle large datasets efficiently without loading the entire dataset into memory. Its zero-copy memory design further enhances speed and memory efficiency.

5.2. Code overview or pseudocode of techniques applied

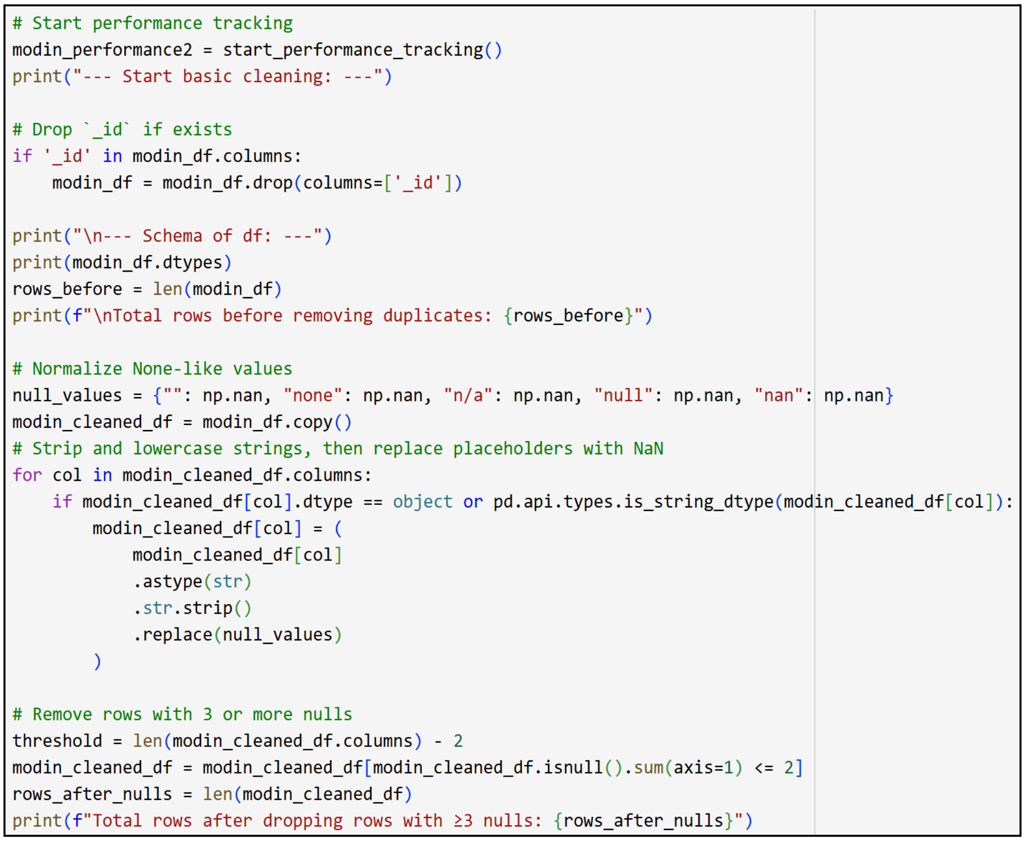
5.2.1. PySpark



```
from pyspark.sql.functions import col, regexp_extract, when, lit, concat, trim, regexp_replace, isnan,

# Start performance tracking
spark_performance2 = start_performance_tracking()
print("\n--- Start basic cleaning: ---")
spark_df = spark_df.drop("_id")
print("\n--- Schema of spark_df: ---")
spark_df.printSchema()
total_rows_before = spark_df.count()
print(f"Total rows before: {total_rows_before}")

# Remove rows with 3 or more nulls
for c in spark_df.columns:
    spark_df = spark_df.withColumn(c, when(isnan(col(c)), None).otherwise(col(c)))
spark_cleaned_df = spark_df.na.drop(thresh=len(spark_df.columns) - 2)
total_rows_after = spark_cleaned_df.count()
print(f"\nTotal rows after removing rows with 3 or more nulls : {total_rows_after}")

# Fill NA values
spark_cleaned_df = spark_cleaned_df.na.fill({"brand": "Unknown",
                 "category": "Unknown",
                 "condition": "Unknown",
                 "shippingfee": 0.0,
                 "link": "Unknown",
                 "price": 0.0,
                 "title": "Unkown"})

# Drop duplicates based on specified columns
spark_cleaned_df = spark_cleaned_df.dropDuplicates(["title","brand","price","shippingfee","condition"])
total_rows_after = spark_cleaned_df.count()
print(f"\nTotal rows after removing duplicates: {total_rows_after}")

# Trim whitespace from string columns
for c in spark_cleaned_df.columns:
    spark_cleaned_df = spark_cleaned_df.withColumn(c, trim(col(c)))
```

Figure 5.2.1: Lazy evaluation in PySpark enabling optimized execution plans and efficient distributed data processing

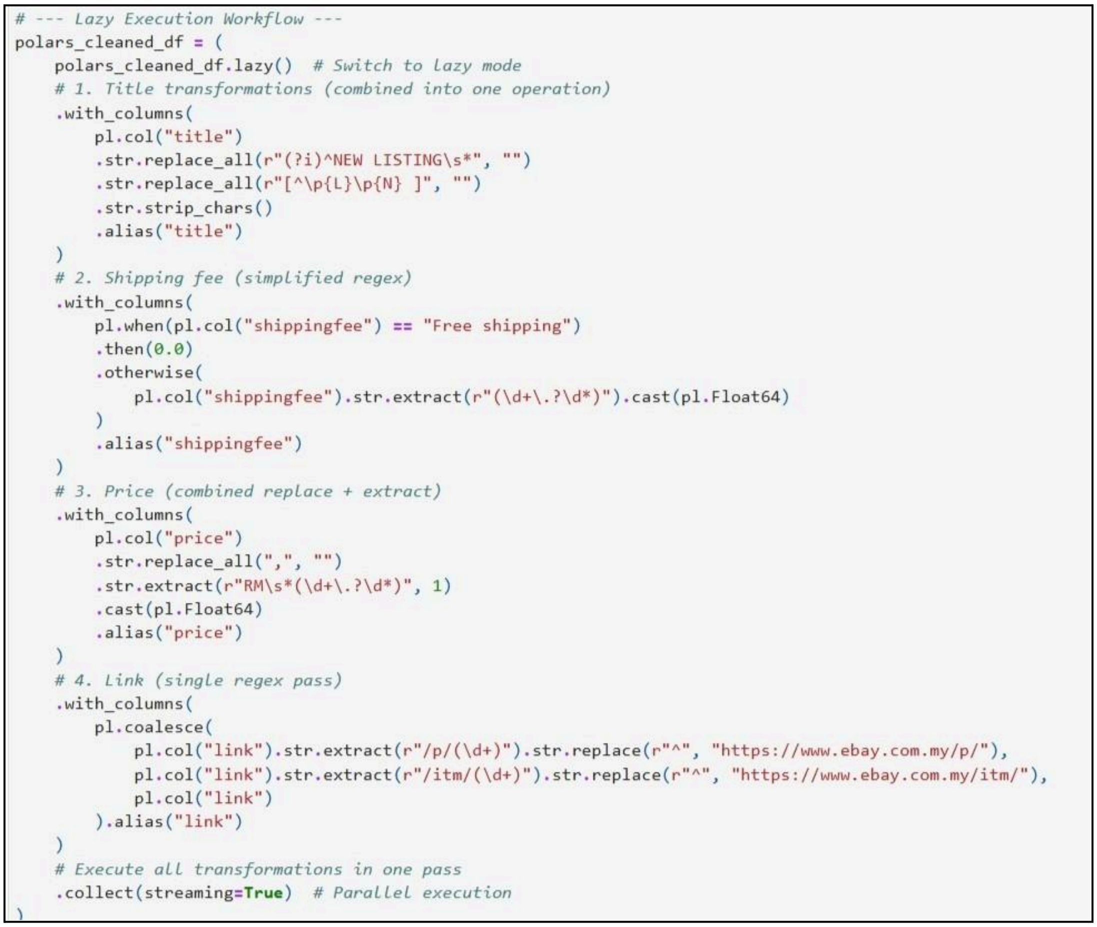
5.2.2. Modin



```
# Start performance tracking
modin_performance2 = start_performance_tracking()
print("--- Start basic cleaning: ---")

# Drop `_id` if exists
if '_id' in modin_df.columns:
    modin_df = modin_df.drop(columns=['_id'])

print("\n--- Schema of df: ---")
print(modin_df.dtypes)
rows_before = len(modin_df)
print(f"\nTotal rows before removing duplicates: {rows_before}")

# Normalize None-like values
null_values = {"": np.nan, "none": np.nan, "n/a": np.nan, "null": np.nan, "nan": np.nan}
modin_cleaned_df = modin_df.copy()
# Strip and lowercase strings, then replace placeholders with NaN
for col in modin_cleaned_df.columns:
    if modin_cleaned_df[col].dtype == object or pd.api.types.is_string_dtype(modin_cleaned_df[col]):
        modin_cleaned_df[col] = (
            modin_cleaned_df[col]
            .astype(str)
            .str.strip()
            .replace(null_values)
        )

# Remove rows with 3 or more nulls
threshold = len(modin_cleaned_df.columns) - 2
modin_cleaned_df = modin_cleaned_df[modin_cleaned_df.isnull().sum(axis=1) <= 2]
rows_after_nulls = len(modin_cleaned_df)
print(f"Total rows after dropping rows with ≥3 nulls: {rows_after_nulls}")
```

Figure 5.2.2: Modin's parallelized execution of pandas operations using Ray or Dask for scalable, efficient multi-core data processing

5.2.3. Polars



```
# --- Lazy Execution Workflow ---
polars_cleaned_df = (
    polars_cleaned_df.lazy()  # Switch to lazy mode
    # 1. Title transformations (combined into one operation)
    .with_columns(
        pl.col("title")
        .str.replace_all(r"(?i)^NEW LISTING\s*", "")
        .str.replace_all(r"[^\p{L}\p{N} ]", "")
        .str.strip_chars()
        .alias("title")
    )
    # 2. Shipping fee (simplified regex)
    .with_columns(
        pl.when(pl.col("shippingfee") == "Free shipping")
        .then(0.0)
        .otherwise(
            pl.col("shippingfee").str.extract(r"(\d+\.?\d*)").cast(pl.Float64)
        )
        .alias("shippingfee")
    )
    # 3. Price (combined replace + extract)
    .with_columns(
        pl.col("price")
        .str.replace_all(",", "")
        .str.extract(r"RM\s*(\d+\.?\d*)", 1)
        .cast(pl.Float64)
        .alias("price")
    )
    # 4. Link (single regex pass)
    .with_columns(
        pl.coalesce(
            pl.col("link").str.extract(r"/p/(\d+)").str.replace(r"^", "https://www.ebay.com.my/p/"),
            pl.col("link").str.extract(r"/itm/(\d+)").str.replace(r"^", "https://www.ebay.com.my/itm/"),
            pl.col("link")
        ).alias("link")
    )
    # Execute all transformations in one pass
    .collect(streaming=True)  # Parallel execution
)
```

Figure 5.2.3: Lazy evaluation in Polars with streaming for efficient parallel processing

6. Performance Evaluation

6.1. Before vs after optimization

Initially, for this project, we utilize Pandas to perform the fundamental data processing tasks: data loading, basic cleaning, and data transformation. To optimize performance and achieve faster execution for these same operations, we then employ three distinct libraries as specialized optimization techniques: Pyspark, leveraging its distributed computing capabilities; Modin, which aims to parallelize existing Pandas workflows with minimal code alteration; and Polars, a modern library built on a Rust backend for efficient, multi-threaded processing.

6.2. Comparison of execution time, peak memory usage, CPU usage and throughput

Operations	Metrics	Libraries			
		Pandas	Pyspark	Modin	Polars
Data Loading	Total Execution Time (s)	12.86	22.43	18.21	13.43
	CPU usage (%)	8.91	16.14	10.11	10.95
	Peak Memory Usage (MB)	508.68	818.06	1211.25	1007.21
	Throughput (records/s)	9843.63	5641.18	6950.08	9423.28
Basic Cleaning	Total Execution Time (s)	2.38	24.52	8.57	0.86
	CPU usage (%)	46.18	2.63	12.84	42.06
	Peak Memory Usage (MB)	555.21	818.09	1224.61	1220.97
	Throughput (records/s)	45031.67	4374.18	12521.91	125322.79
Data Transformation	Total Execution Time (s)	2.39	21.16	4.39	0.17
	CPU usage (%)	27.04	2.65	18.10	47.74
	Peak Memory Usage (MB)	580.05	818.13	1224.61	1197.61
	Throughput (records/s)	44976.16	5068.56	24420.35	640177.27

Table 6.2.1 : Comparison between 4 Libraries

Conclusion

- **Polars:** Competitive in loading. Excelled in basic cleaning and data transformation, delivering unmatched speed and throughput, making it the most efficient library for these tasks.

- **Pandas:** Achieved the fastest data loading speed. However, it was significantly slower than Polars for basic cleaning and data transformation tasks, with much lower throughput.

- **Modin:** Slower than Pandas and Polars for loading and basic cleaning. Its data transformation speed also trailed both, offering limited performance benefits.

- **PySpark:** Consistently exhibited the slowest performance. It had the longest execution times and lowest throughput for data loading, basic cleaning, and data transformation tasks.

6.3. Charts and graphs

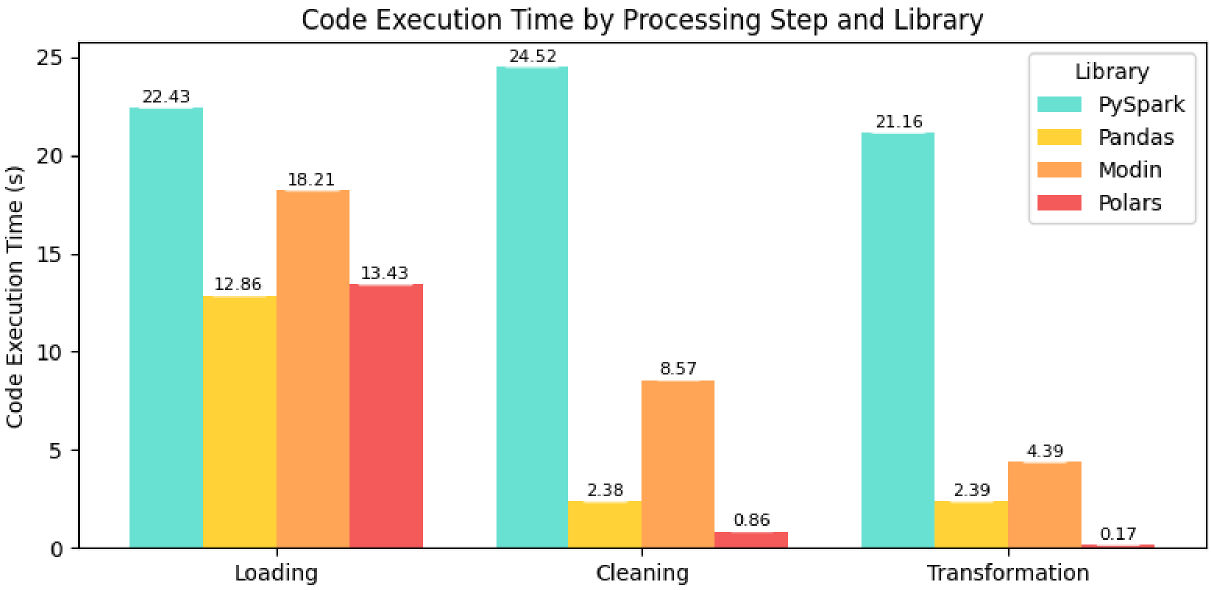


Figure 6.3.1 : Bar Chart for Code Execution Time

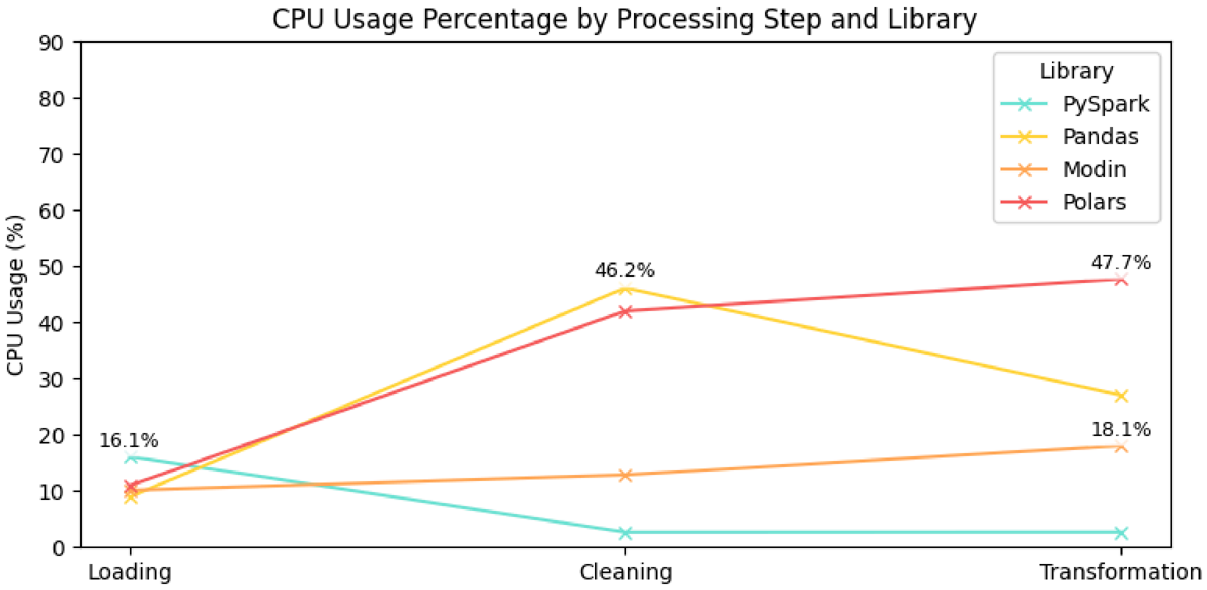


Figure 6.3.2 : Line Graph for CPU Usage Percentage (%)

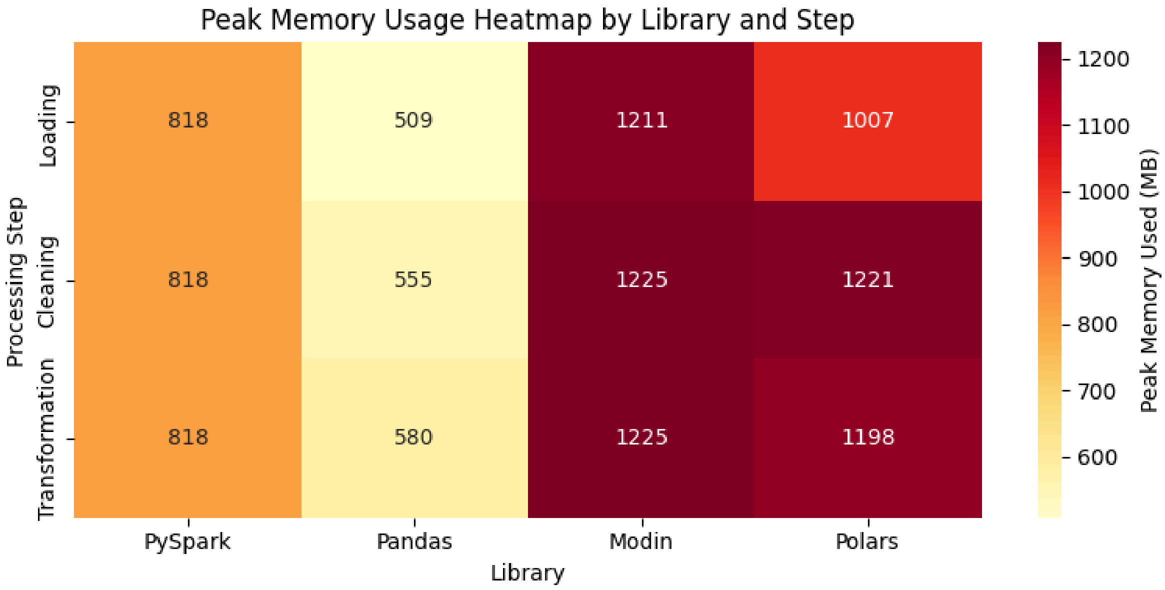


Figure 6.3.3 : Heatmap for Peak Memory Usage (MB)

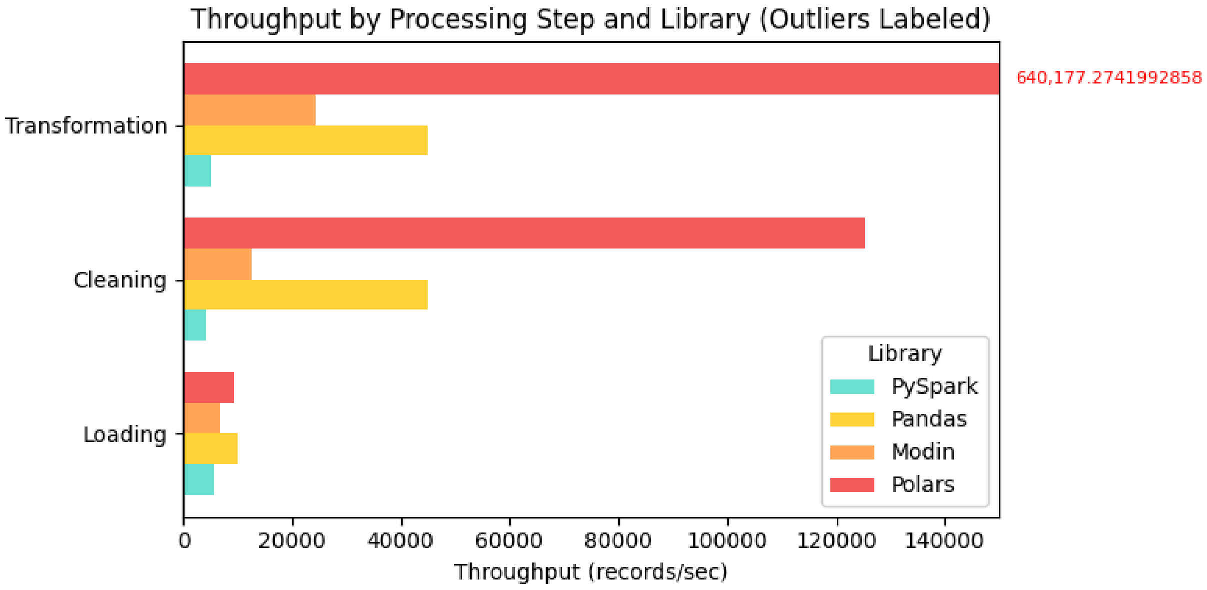


Figure 6.3.4 : Horizontal Bar Chart for Throughput (records/s)

7. Challenges & Limitations

7.1. What didn't go as planned

Despite careful planning, several unexpected challenges emerged during the data scraping and processing stages. These issues affected data quality, processing efficiency, and overall project accuracy. The following points summarize the key obstacles encountered:

- **Inconsistent Website Behavior:** Different libraries may experience different levels of reliability in retrieving complete data during web scraping. For example, sometimes some pages may fail to load using Selenium or Playwright due to bot detection on eBay. Such conditions affected the scraping accuracy and completeness.

- **Rate Limiting and Fake Results:** eBay only allows loading up to 169 pages or around 10,000 results per search. Beyond that, the platform may return repeated or fake listings, making it difficult to retrieve comprehensive datasets. This introduced data duplication and skewed results.

- **Varying HTML Structures:** In eBay, the HTML structure for similar fields was not uniform across items. For instance, some shipping fees were inside a

<span> while others were within a <div> with class <s-item>. This led to massive null values during scraping and required additional conditional parsing logic to handle them.

- **Data Processing Library Issues:** Initially, Dask was used for data processing, but it performed poorly with datasets larger than 100k rows. It took extremely long or failed to complete when doing data transformation. After further research, Dask was replaced by Modin, which showed better performance in this scenario.

- **Inconsistent Null Handling Across Libraries:** Data processing across Polars, Pandas, PySpark, and Modin were more complex than expected. Since each library handles null values and missing data differently, the results when applying the same logic were affected. For example, Polars treats nulls more strictly, while PySpark may preserve them unless explicitly filtered. This required writing library-specific data cleaning code to ensure consistency.

7.2. Any limitations of your solution

While the solution achieved its primary objectives, several limitations were identified that could impact its scalability, efficiency, and completeness. These constraints highlight areas for future improvement and should be considered when extending or deploying the solution in other contexts:

- **Single-Machine Execution:** Some of the data processing libraries used like PySpark and Modin are designed for distributed computing environments. However, all testing in this project was done on a single machine. This limits their ability to perform true scalability and high-performance processing.

- **Processing Overhead:** Libraries like PySpark take extra time and system resources to initialize before running any operations. For medium-sized datasets (approximately 100,000 records), this setup time slowed down the workflow compared to more lightweight tools like Pandas or Polars.

- **No Real-Time Data Handling:** All data scraping and processing were done in batch mode. It means that we scraped everything first, then processed it. This makes the solution unsuitable for real-time use cases like live product tracking or continuous updates.

- **Possible Missed Data Due to Complex Web Features:** Although we added logic to handle dynamic elements like the shipping fee, we still couldn’t manually verify every item. This means that there is a chance that some values were still missed or recorded as null, especially for listings with unusual layouts or hidden content.

8. Conclusion & Future Work

8.1. Summary of findings

The evaluation tested the performance of Pandas, PySpark, Polars, and Modin using total execution time, CPU usage, memory utilization, and how fast these libraries handle each of three key operations. loading the data, basic cleaning of the data, and transforming the data.

Framework	Strengths	Use Case Fit
Pandas	- Very fast in light workloads (Basic Cleaning and Transformation) - High CPU utilization leads to efficient single-threaded performance - Excellent throughput for smaller datasets	- Ideal for small to medium-sized datasets - Best when running on a single machine - Great for quick prototyping and EDA
Pyspark	- Handles large datasets efficiently - Distributed processing across clusters - Stable memory usage across tasks	- Best fit for big data scenarios - Suitable for cluster or cloud environments - Useful when working with very large-scale ETL pipelines
Polars	- Fastest execution across all tasks - Highest throughput, especially in transformation tasks - Efficient multi-threaded processing - Balanced CPU and memory usage	- Excellent for performance-critical tasks - Suitable for real-time or near real-time processing - Best when performance and scalability are top priorities
Modin	- Parallelized execution with Pandas-like syntax - Better than Pandas in some moderate workloads - Moderate CPU and memory usage	- Good for users familiar with Pandas but needing faster processing - Fits medium-sized data on multi-core machines - Useful for scaling up existing Pandas code without major refactoring

8.2. What could be improved

Although each framework is useful in its own way, there are situations where their limits play a role. Understanding of such areas allows one to choose suitable tools for the right task .

1. Better Resource Utilization in PySpark

Despite being designed for large-scale data processing, PySpark showed very low CPU usage during basic tasks like cleaning and transformation. This underutilization suggests that PySpark is inefficient for small to medium-sized datasets and may introduce unnecessary overhead. Improving how PySpark handles lighter workloads or offering a lightweight mode could make it more versatile.

2. More Consistent Speed Gains in Modin

Modin aims to speed up Pandas by enabling parallel processing, but the results show inconsistent performance gains across different operations. For example, while it improves transformation speed compared to Pandas, it still lags behind Polars. This suggests room for optimization in how Modin distributes and executes tasks internally, especially in basic cleaning.

3. Expanded Feature Support in Polars

Polars delivered the best throughput and speed across nearly all tasks, especially transformations. However, it’s still a relatively new library and doesn’t yet match Pandas in terms of ecosystem or advanced functionality. Improving compatibility with existing Python tools and adding support for more complex operations could help Polars gain broader adoption.

4. Enhanced Scalability in Pandas

Pandas performed well in smaller tasks with high throughput and efficiency, but it doesn't scale well with growing data sizes. Memory usage and single-threaded processing become limitations for larger workloads. Incorporating built-in support for parallelism or native scaling features could significantly extend its usability.

9. References

[1] Python Software Foundation, "Python Language Reference, version 3.11", [Online]. Available: https://www.python.org

[2] The Pandas Development Team, "Pandas Documentation", [Online]. Available: https://pandas.pydata.org/docs/

[3] MongoDB Inc., "MongoDB Manual", [Online]. Available: https://www.mongodb.com/docs/manual/

[4] PyMongo Developers, "PyMongo Documentation", [Online]. Available: https://pymongo.readthedocs.io/en/stable/

[5] NumPy Developers, "NumPy Documentation", [Online]. Available: https://numpy.org/doc/

[6] Python Software Foundation, "re — Regular Expression Operations", [Online]. Available: https://docs.python.org/3/library/re.html

[7] eBay Inc., "eBay Malaysia - Electronics, Cars, Fashion, Collectibles & More", [Online]. Available: https://www.ebay.com.my

[8] Polars Developers, "Polars User Guide", [Online]. Available: https://pola-rs.github.io/polars/

[9] Apache Software Foundation, "PySpark Documentation", [Online]. Available: https://spark.apache.org/docs/latest/api/python/

[10] Modin Contributors, "Modin: Speed up your Pandas workflows by changing a single line of code", [Online]. Available: https://modin.readthedocs.io/en/latest/

[11] Scrapy Developers, "Scrapy Documentation", [Online]. Available: https://docs.scrapy.org/en/latest/

[12] Apache Software Foundation, "Apache Spark Documentation", [Online]. Available: https://spark.apache.org/docs/latest/

[13] L. Richardson et al., "BeautifulSoup: Python library for pulling data out of HTML and XML files", [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[14] PSUtil Developers, "psutil – Process and System Utilities", [Online]. Available: https://psutil.readthedocs.io/en/latest/

[15] R. Mitchell, Web Scraping with Python: Collecting More Data from the Modern Web, 2nd ed., O'Reilly Media, 2018.

[16] M. Lutz, Learning Python, 5th ed., Sebastopol, CA: O'Reilly Media, 2013.

10. Appendices

10.1. Sample code snippets

- Web Crawler (Selenium)

```
# Base URL with pagination and price filters
```

```
base_url = "https://www.ebay.com.my/b/Consumer-Electronics/293/bn_1865552?_pgn={}&_udlo=65&_udhi=150"
max_pages = 169
```

```
# Loop through pages
```

```
for page_num in range(1, max_pages + 1):
    url = base_url.format(page_num)
    driver.get(url)
    time.sleep(2 + random.uniform(0.5, 1.5))  # Rate limiting

    listings = driver.find_elements(By.CSS_SELECTOR, "div.brwrvr__item-card__signals")

    for listing in listings:
        try:
            title = listing.find_element(By.CSS_SELECTOR, 'h3.textual-display.bsig__title__text').text
        except NoSuchElementException:
            title = None

        try:
            price = listing.find_element(By.CSS_SELECTOR, 'span.bsig__price--displayprice').text
        except NoSuchElementException:
            price = None

        try:
            shippingfee = listing.find_element(By.CSS_SELECTOR, 'span.bsig__logisticsCost').text
        except NoSuchElementException:
            shippingfee = None

        try:
            conditions = listing.find_elements(By.CSS_SELECTOR,
'span.bsig__listingCondition > span')
            texts = [c.text.strip() for c in conditions if
c.text.strip() and c.text.strip() != "·"]
            condition = texts[0] if len(texts) > 0 else None
            brand = texts[1] if len(texts) > 1 else None
        except NoSuchElementException:
            condition = None
            brand = None

        try:
            link = listing.find_element(By.CSS_SELECTOR,
'a').get_attribute('href')
        except NoSuchElementException:
            link = None
```

```
 # Output the full data for each item
```

```
print({
            "title": title,
            "price": price,
            "shippingfee": shippingfee,
            "condition": condition,
            "brand": brand,
            "link": link
        })
```

- Data Loading (Pyspark)

```
# Load and Merge Data from Multiple MongoDB Collections
```

```
from pymongo import MongoClient

client = MongoClient("your_mongodb_uri")
db = client["HPDP_eBay"]
all_data = []

collection_names = [
    'eBay_CamerasPhoto',
    'eBay_Collectibles',
    'eBay_ToysHobbies',
    'eBay_ConsumerElectronics'
]

for name in collection_names:
    documents = list(db[name].find())
    for doc in documents:
        doc['_id'] = str(doc['_id'])  # Convert ObjectId for
compatibility
    all_data.extend(documents)

spark_df = spark.createDataFrame(all_data)
```

- Basic Cleaning (Pyspark)

```
# Remove duplicate rows based on key columns
```

```
total_rows_before = spark_df.count()
print(f"Total rows before removing duplicates:
{total_rows_before}")

spark_cleaned_df = spark_df.dropDuplicates(["title", "brand",
"price", "shipping fee", "condition"])

total_rows_after = spark_cleaned_df.count()
print(f"\nTotal rows after removing duplicates:
{total_rows_after}")
```

```
# Drop rows with 3 or more nulls and fill missing values
```

```
spark_cleaned_df = spark_cleaned_df.na.fill({
    "brand": "Unknown",
    "category": "Unknown",
    "condition": "Unknown",
    "shippingfee": 0.0,
    "link": "Unknown",
    "price": 0.0,
    "title": "Unkown"
})
```

- Data Transformation (Pyspark)

```
# Clean title and remove unwanted text
```

```
spark_cleaned_df2 = spark_cleaned_df.withColumn(
    "title",
    regexp_replace(
        regexp_replace("title", r"(?i)^NEW LISTING\s*", ""),
        r"[^\p{L}\p{N} ]", "")
)
```

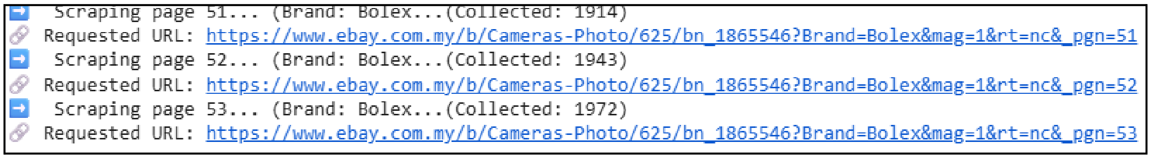
```
# Convert shipping fee: "Free shipping" → 0.00, extract number
otherwise
```

```
spark_cleaned_df2 = spark_cleaned_df2.withColumn(
    "shippingfee",
    when(
        spark_cleaned_df2["shippingfee"] == "Free shipping", 0.00
    ).otherwise(
        regexp_extract(col("shippingfee"), r"(\d+\.\d+|\d+)",
0).cast("double")
    )
)
```

10.2. Screenshots of output

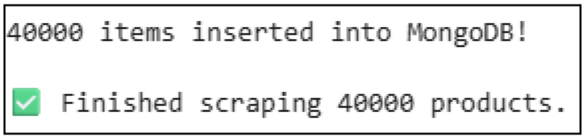
- ### Web Crawler

Printing the current page number, brand, and total items collected after each page



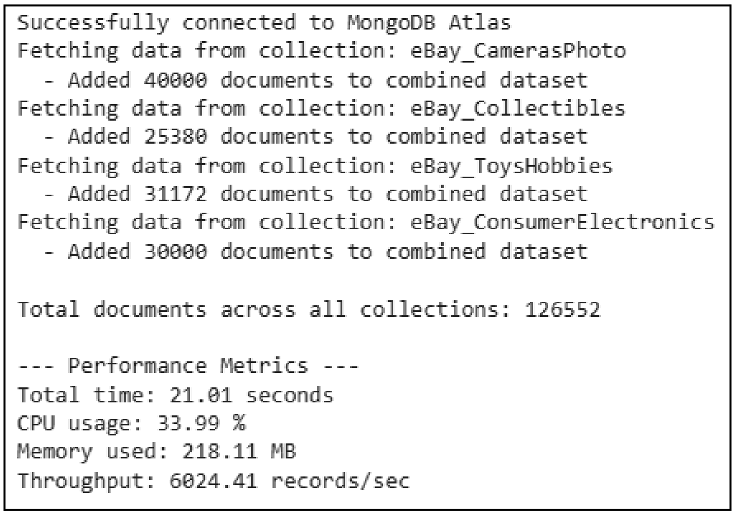
```
➡️  Scraping page 51... (Brand: Bolex...(Collected: 1914)
🔗 Requested URL: https://www.ebay.com.my/b/Cameras-Photo/625/bn_1865546?Brand=Bolex&mag=1&rt=nc&_pgn=51
➡️  Scraping page 52... (Brand: Bolex...(Collected: 1943)
🔗 Requested URL: https://www.ebay.com.my/b/Cameras-Photo/625/bn_1865546?Brand=Bolex&mag=1&rt=nc&_pgn=52
➡️  Scraping page 53... (Brand: Bolex...(Collected: 1972)
🔗 Requested URL: https://www.ebay.com.my/b/Cameras-Photo/625/bn_1865546?Brand=Bolex&mag=1&rt=nc&_pgn=53
```

Finished scraping 40k rows and inserted into mongodb



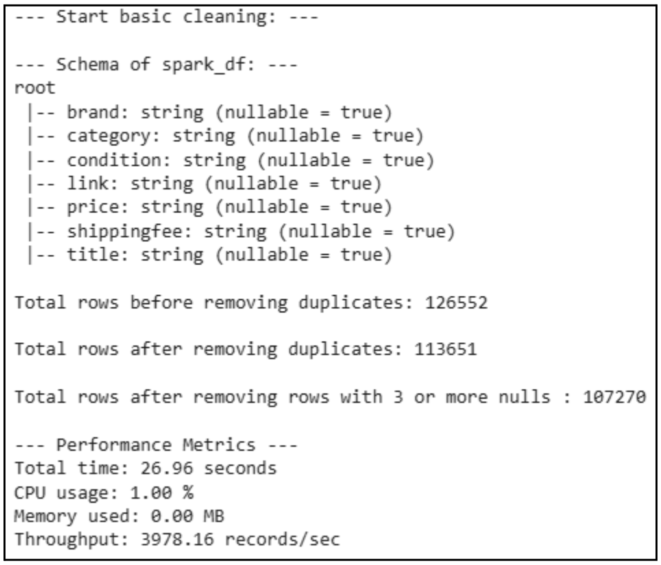
```
40000 items inserted into MongoDB!

✅ Finished scraping 40000 products.
```

- ### Data Load



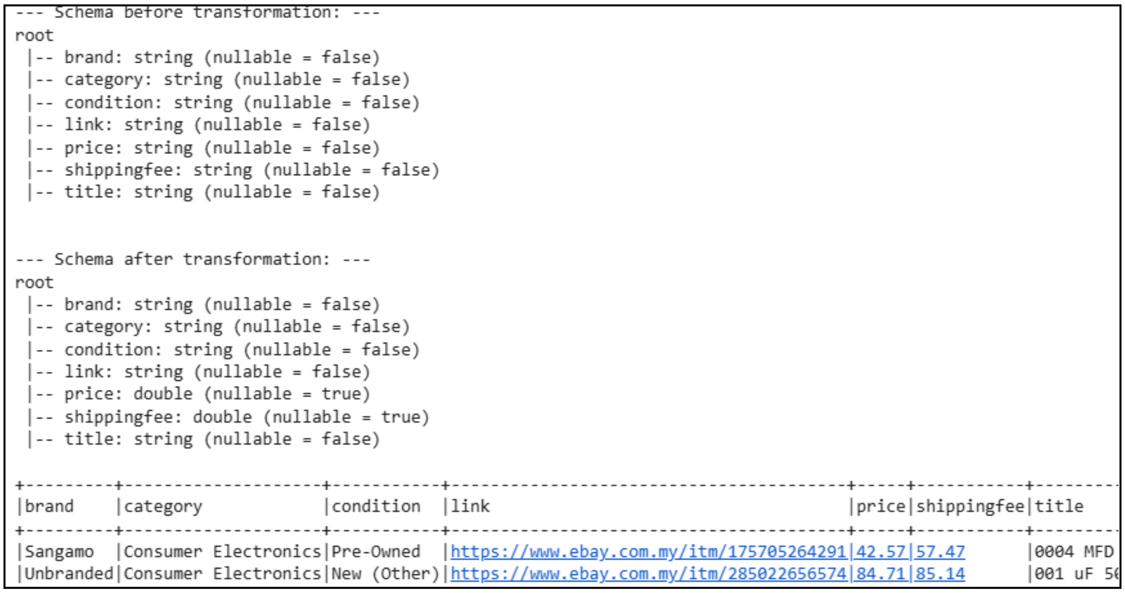
```
Successfully connected to MongoDB Atlas
Fetching data from collection: eBay_CamerasPhoto
  - Added 40000 documents to combined dataset
Fetching data from collection: eBay_Collectibles
  - Added 25380 documents to combined dataset
Fetching data from collection: eBay_ToysHobbies
  - Added 31172 documents to combined dataset
Fetching data from collection: eBay_ConsumerElectronics
  - Added 30000 documents to combined dataset

Total documents across all collections: 126552

--- Performance Metrics ---
Total time: 21.01 seconds
CPU usage: 33.99 %
Memory used: 218.11 MB
Throughput: 6024.41 records/sec
```

- Basic Cleaning



```
--- Start basic cleaning: ---

--- Schema of spark_df: ---
root
 |-- brand: string (nullable = true)
 |-- category: string (nullable = true)
 |-- condition: string (nullable = true)
 |-- link: string (nullable = true)
 |-- price: string (nullable = true)
 |-- shippingfee: string (nullable = true)
 |-- title: string (nullable = true)

Total rows before removing duplicates: 126552

Total rows after removing duplicates: 113651

Total rows after removing rows with 3 or more nulls : 107270

--- Performance Metrics ---
Total time: 26.96 seconds
CPU usage: 1.00 %
Memory used: 0.00 MB
Throughput: 3978.16 records/sec
```

- Data Transformation



```
--- Schema before transformation: ---
root
 |-- brand: string (nullable = false)
 |-- category: string (nullable = false)
 |-- condition: string (nullable = false)
 |-- link: string (nullable = false)
 |-- price: string (nullable = false)
 |-- shippingfee: string (nullable = false)
 |-- title: string (nullable = false)


--- Schema after transformation: ---
root
 |-- brand: string (nullable = false)
 |-- category: string (nullable = false)
 |-- condition: string (nullable = false)
 |-- link: string (nullable = false)
 |-- price: double (nullable = true)
 |-- shippingfee: double (nullable = true)
 |-- title: string (nullable = false)

+---------+-------------------+-----------+----------------------------------------+-----+-----------+--------
|brand    |category           |condition  |link                                    |price|shippingfee|title
+---------+-------------------+-----------+----------------------------------------+-----+-----------+--------
|Sangamo  |Consumer Electronics|Pre-Owned  |https://www.ebay.com.my/itm/175705264291|42.57|57.47      |0004 MFD
|Unbranded|Consumer Electronics|New (Other)|https://www.ebay.com.my/itm/285022656574|84.71|85.14      |001 uF 5
```

10.3. Links to full code repo or dataset

- Github Repositories
- Project Proposal
- System Architecture
- Crawler Source Code
- Data Processing and Optimization technique Source Code