UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING

SECP3133-02 HIGH PERFORMANCE DATA PROCESSING

PROJECT 1 - REPORT

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

PREPARED BY: GROUP 3

NAME	MATRIC NO.
MUHAMMAD DANIEL HAKIM BIN SYAHRULNIZAM	A22EC0207
NICOLE LIM TZE YEE	A22EC0123
NUR ALEYSHA QURRATU'AINI BINTI MAT SALLEH	A22EC0241
WONG KHAI SHIAN NICHOLAS	A22EC0292

PREPARED FOR: DR. ARYATI BINTI BAKRI

DATE: 16/05/2025

Table of Contents

1.0 Introduction

1.1 Background of the Project

In the era of big data, web-based information has become a crucial asset for a wide range of industries and research fields. Websites, especially news portals, generate vast amounts of dynamic and continuously updated data that can offer valuable insights when collected and analyzed effectively. However, the process of gathering this data at scale introduces several technical challenges such as dynamic page rendering, data redundancy, ethical scraping, and performance bottlenecks during processing.

High-Performance Computing (HPC) techniques offer solutions to these challenges by improving the efficiency, scalability, and reliability of web crawling systems. Techniques such as multithreading, multiprocessing, and distributed computing allow data engineers to handle large volumes of web data within reasonable time and resource constraints.

This project focuses on designing and implementing a high-performance data collection and processing pipeline through a web crawler. The crawler is optimized using HPC techniques to collect structured data efficiently and effectively from a Malaysian website.

1.2 Objectives

The main objectives of this project are as follows:

- To develop a robust web crawler capable of extracting a minimum of 100,000 structured records from a Malaysian website.
- To apply high-performance computing techniques, such as multithreading, multiprocessing, and asynchronous requests, to enhance the crawler's performance.
- To clean, transform, and store the collected data in a structured format suitable for downstream applications or analysis.
- To evaluate the performance of the system before and after optimization using measurable performance metrics such as execution time, CPU usage, memory consumption, and throughput.

- To work collaboratively in a diverse team environment and produce a final report, source code, structured dataset, and a performance evaluation.

1.3 Target Website and Data to Be Extracted

The selected data source for this project is Utusan Malaysia (https://www.utusan.com.my/), a prominent and long-standing Malaysian news organization with a significant online presence. Utusan Malaysia publishes a wide array of news articles covering various categories, including national news, the economy, the Malaysian Ringgit, international news, sports, features, entertainment, and politics.

The specific data fields targeted for extraction are as follows:

- Article Title: The main title or headline of the published news article.
- Publication Date: The precise date on which the news article was published online.
- Article Category: The designated section or category under which the news article is classified on the website (e.g., Nasional, Ekonomi, Sukan).
- Article URL: The unique Uniform Resource Locator (web address) that links directly to the full content of the article.

To achieve the project's data acquisition target, the developed web crawler will be designed to navigate across multiple pages of the Utusan Malaysia website, systematically parse the HTML structure (and potentially utilize Selenium for dynamic content) to extract the relevant information for each identified article within the responsible sections. The extracted and cleaned records will be stored in a standardized Comma Separated Values (CSV) file format. Throughout the scraping process, we will adhere to ethical best practices by diligently respecting the website's robots.txt policy and implementing appropriate rate-limiting and automated retry mechanisms to prevent server overload and ensure responsible data collection.

The team will focus on the following sections based on individual responsibilities:

- Aleysha - Nasional
- Nicole - Ekonomi, Ringgit, Luar Negara
- Daniel - Sukan, Rencana, Pancaindera
- Nicholas - Gaya, Komuniti

2.0 System Design & Architecture

2.1 Description of Architecture

The architecture of our high-performance web scraping system for Utusan Malaysia follows a distinct multi-stage process, emphasizing a centralized storage for raw data before cleaning and optimization. The system's components and data flow are illustrated in the diagram below:

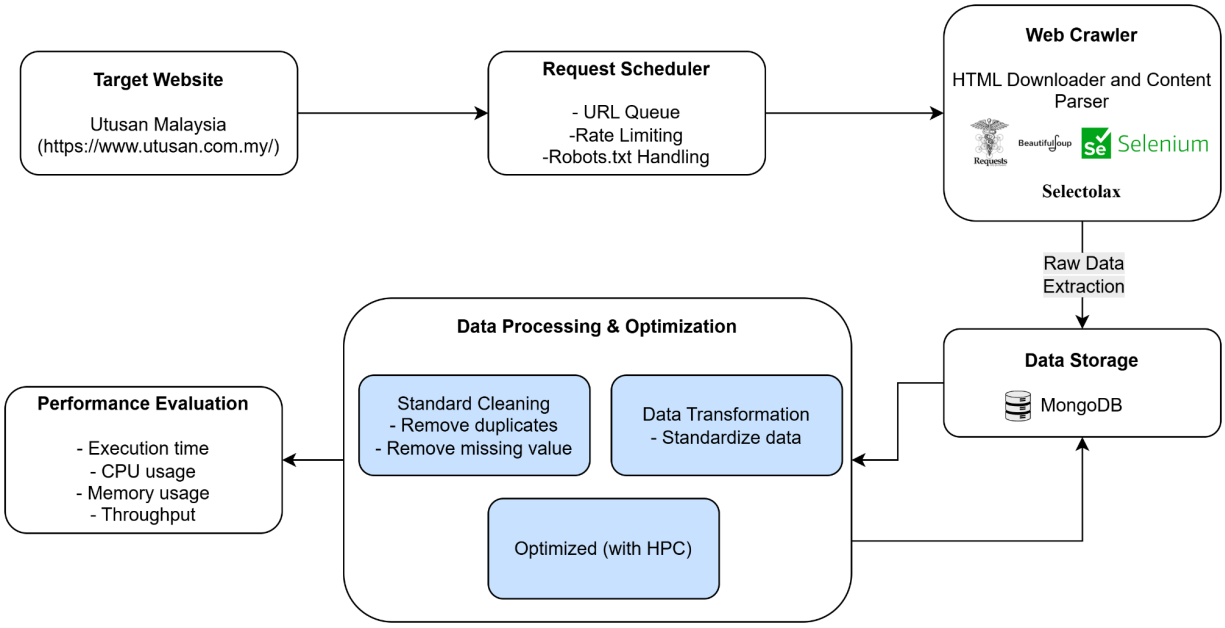


Figure 2.1: System Architecture

As planned, the process begins with the target website, Utusan Malaysia (https://www.utusan.com.my/). The Request Scheduler manages the URLs to be crawled for the various sections assigned to each team member.

Our crawler configurations are precisely defined to target these specific sections and subcategories to ensure comprehensive and non-overlapping data collection. The web crawler modules then fetch content from their designated sections, employing the most suitable scraping libraries:

- Crawler 1, designated for a focused subset of the Nasional section, specifically targets the subcategories of Jenayah (Crime), Tragedi (Tragedies), and Mahkamah (Courts), utilizing

Playwright to handle dynamically loaded web pages and Beautiful Soup for parsing and extracting the desired information.

- Crawler 2 is responsible for the Ekonomi, Ringgit, and Luar Negara sections, employing Requests and Beautiful Soup for general content retrieval while also incorporating Selenium to handle dynamically loaded elements. Section Ekonomi includes subcategories such as Hartanah (Property), Kewangan (Finance), Korporat (Corporate), Produk (Products), and Usahawan (Entrepreneurs). The Luar Negara section is further divided into geographical subcategories like Afrika (Africa), Amerika (America), Asia, Asia Barat (West Asia), Asia Tenggara (Southeast Asia), and Eropah (Europe).
- Crawler 3 is dedicated to the Sukan, Rencana, and Pancaindera sections, leveraging the speed and efficiency of Selectolax for parsing. The Sukan section includes subcategories like Bola Sepak (Football), Badminton, E-Sukan (E-Sports), Lumba Basikal (Cycling), Lumba Kereta (Car Racing), Lumba Motor (Motorcycle Racing), Sepak Takraw, and Semua Sukan (All Sports), while Rencana includes Wawancara (Interviews) and Surat Pembaca (Letters to the Editor).
- Finally, Crawler 4 is designed for the Gaya and Komuniti (subcategory of Nasional) sections, utilizing Requests and Beautiful Soup, with Selenium integrated to manage any dynamically rendered content. The Gaya section comprises subcategories such as Agama (Religion), Agro (Agriculture), Anakku (My Child), Deko (Decor), Fesyen (Fashion), Hiburan (Entertainment), Keluarga/Wanita (Family/Women), Kesihatan (Health), and Pendidikan (Education).

The raw data extraction from these modules results in an initial, uncleaned dataset. This uncleaned data is then directly loaded into data storage. MongoDB is chosen at this stage for its flexibility in handling potentially varied and semi-structured raw data as it is initially scraped from different website sections.

Following the initial data ingestion, the process diverges into data processing and optimization. This stage involves reading the uncleaned data from MongoDB and applying various cleaning and optimization techniques. Critically, we will perform data cleaning in two distinct ways:

1. Standard Cleaning and Data Transformation: Applying a baseline set of cleaning and transformation procedures.
2. Optimized Cleaning: Applying the same cleaning procedures but leveraging High-Performance Computing (HPC) techniques like multithreading or multiprocessing to accelerate the cleaning process.

Both the standard cleaned data and the optimized cleaned data are then loaded back into Data Storage (MongoDB). This allows us to compare the performance benefits of applying HPC during the data cleaning phase.

In order to evaluate the effectiveness of optimization, performance metrics are recorded, which include execution time, CPU usage, memory usage, and throughput. The results are visualized through charts and graphs to provide insights into how HPC methods improve performance.

2.2 Tools and Frameworks Used

Our project utilizes a range of tools and frameworks to facilitate efficient web scraping.

Table 2.2.1 Libraries Used and The Descriptions

Library	Description
Request	A simple and elegant HTTP library for Python, used to send HTTP requests to fetch web pages.
BeautifulSoup	Easy-to-use HTML parser; good for static HTML; simple syntax for extracting elements by tag/class.
Selenium	Needed when content loads dynamically or requires interaction (e.g., click to expand menu or paginate).
Selectolax	Perfect for scraping many static pages quickly and repeatedly.
Asyncio	Allows multiple page loads and content parsing to occur without blocking the main thread.

2.3 Roles of Team Members

Roles were assigned to ensure smooth and organized workflow.

Table 2.3.1 Roles of Team Members

Team Member	Role	Task Distribution
Nicholas	Team Leader	● Choose a Malaysian Website ● Web crawler library used: Selenium ● Optimization library used: Swifter
Nur Aleysha	Web Crawler Developer	● Web crawler library used: BeautifulSoup + Asyncio ● Create and load raw and clean data into MongoDB ● Optimization library used: Dask
Nicole	Data Handler	● Combine all scraped data to load in MongoDB ● Web crawler library used: BeautifulSoup + Selenium ● Optimization library used: Modin
Daniel Hakim	Technical Support & Tester	● Web crawler library used: Selectolax ● Optimization library used: Polars

3.0 Data Collection

3.1 Crawling Method

The data collection process involved a focused web crawling approach to gather information from the Utusan Malaysia website. To systematically navigate the website's structure, a combination of URL pattern manipulation and direct URL access was employed. The base URL for the website is https://www.utusan.com.my. For sections organized with pagination, the crawling process utilized a consistent URL structure: https://www.utusan.com.my/category/[category_name]/[subcategory_name]/page/[page_no]. This pattern allowed the crawler to iterate through multiple pages within a given subcategory by incrementing the page_no parameter, as demonstrated by the example URL: https://www.utusan.com.my/category/ringgit/hartanah/page/2/.

Conversely, certain sections of the website, specifically 'Ringgit' and 'Pancaindera', did not employ pagination. For these sections, the data was retrieved directly from the base URL associated with the category name: https://www.utusan.com.my/[category_name], as exemplified by the URL: https://www.utusan.com.my/pancaindera/. This approach ensured that all available data within these non-paginated sections was captured efficiently.

The crawler was designed to extract the following data fields from the HTML structure of the web pages:

- **Article Title:** The title of the article was extracted by locating the <h3> HTML element with the class name 'jeg_post_title'. The text content within this element, after removing any leading or trailing whitespace, was then captured as the article title.
- **Publication Date:** The publication date was obtained by identifying the element with the class name 'jeg_meta_date'. The text content within this element, after removing any leading or trailing whitespace, was extracted.
- **Article Category:** The category of the article was derived from the URL structure itself. The [category_name] part of the URL was used to assign the article to its respective category.

- **Article URL:** The full URL of the article was extracted by locating the <a> tag within the <h3> element (identified for the title). The 'href' attribute of this anchor tag provided the complete URL.

3.2 Number of Records Collected

Table 3.2.1 Numbers of Records Collected

Crawler	Records Collected
Crawler 1	28544
Crawler 2	48847
Crawler 3	42450
Crawler 4	41854
Total	161695

3.3 Ethical Considerations

To ensure responsible data collection, the web scraper was developed following ethical guidelines:

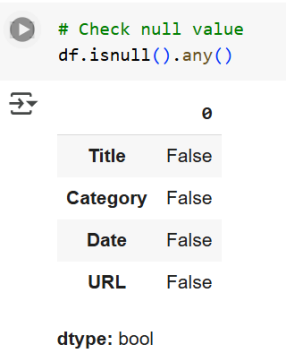
- The target website, Utusan Malaysia, is a public news portal, and only data that was openly accessible was collected. The web scraping process strictly adhered to the website's robots.txt to avoid accessing restricting areas, adhering to the site usage policy.
- No private data or personal information was collected or stored. The content collected consisted only of article metadata like titles, publish dates, categories, and URLs which were already publicly accessible from the publisher.
- The scrapers were set with proper rate-limiting and delays to avoid putting too much load on the host server. The approach reduced opportunities for service interruption or damage to the operationality of the website.
- The data collected was solely utilized for research and academic purposes for the undertaking of the project. There is no redistribution of the scraped content as intended.

4.0 Data Processing

4.1 Cleaning Methods

The collected data underwent several cleaning steps to ensure its quality and consistency, processing the data into a usable form for subsequent data analysis. These steps included:

- Handling null values: Null or missing values within the dataset were addressed to prevent errors or biases in later analysis.

Python Code	Explanation
`df.isnull().any()`	The code is used to verify the presence of null values within the dataset. The result returned 'True' if any column contained null values. Based on the results, the dataset does not contain any null values. 
`df2 = df2[['Title', 'Category', 'Date', 'URL']].dropna()`	The line of code removes rows containing null values from the DataFrame.

- Removing duplicate records:

Python Code	Explanation
`df2['URL'] = df2['URL'].apply(lam`	The code removes any leading or trailing spaces from URL strings to standardize the data. Then, it eliminates

`bda x: str(x).strip() if isinstance(x, str) else x)` `df2.drop_duplicates(subset='URL', keep='first', inplace=True)`	duplicate rows based on the 'URL' column, keeping only the first occurrence of each unique URL. This step prevents skewed analysis results due to repeated data.

4.2 Transformation and Formatting

The collected data also underwent transformation to ensure its quality and consistency, processing the data into a usable form for subsequent data analysis. These steps included:

- Standardizing the 'Category' column:

Python Code	Explanation
`df2['Category'] = df2['Category'].astype(str).str.capitalize()`	The code standardizes the 'Category' column by first converting all values to strings and then capitalizing the first letter of each category while lowercasing the rest (e.g. “Jenayah”, “Ekonomi”). This ensures consistency and avoids issues caused by variations in capitalization.

- Removing rows with 'No Date' values:

Python Code	Explanation
`df2 = df2[~df2['Date'].astype(str).str.strip().str.lower().eq('no date')]`	This code filters out rows with "no date" in the 'Date' column. It converts the column to strings, removes extra spaces, makes it lowercase, and then filters to keep only rows where the date is not "no date". This is because such entries lack essential temporal information, making them unsuitable for time-based analysis.

- Aligning the date format:

Python Code	Explanation
	This code standardizes date strings. It uses a dictionary (month_map) to convert month names (in Malay) to numbers and a function (convert_date) to reformat dates to "YYYY-MM-DD HH:MM:00". The function parses the date string using a regular expression, handles AM/PM, and includes error handling. This standardization ensures that dates are represented consistently, allowing for accurate sorting, filtering, and time-series analysis.

4.3 Data Structure

After completing the data cleaning processes using five different libraries (Pure Python, Polars, Modin, Dask, and Swifter), the resulting cleaned datasets were exported into CSV format and uploaded into separate collections in MongoDB for structured storage and future retrieval. The raw datasets were also uploaded into a collection in MongoDB before cleaning (Raw_Data).

Each cleaned dataset maintains a consistent structure with the following fields:

Field Name	Data Type	Description
Title	String	News article title
Category	String	Section or category (e.g., Ekonomi, Agama) – standardized with capitalization

Date	Datetime	Publication date – cleaned and standardized to YYYY-MM-DD HH:MM:SS format
URL	String	Unique article link

Storage Details:

- CSV Files: Each library's cleaned dataset is exported as a CSV file (e.g., cleaned_polars.csv, cleaned_modin.csv) for comparison and external use.

- MongoDB Collections: Cleaned data is also stored in MongoDB under the database webcrawler_project, with separate collections per library:

    - cleaned_python_data
    - cleaned_polars_data
    - cleaned_modin_data
    - cleaned_dask_data
    - cleaned_swifter_data

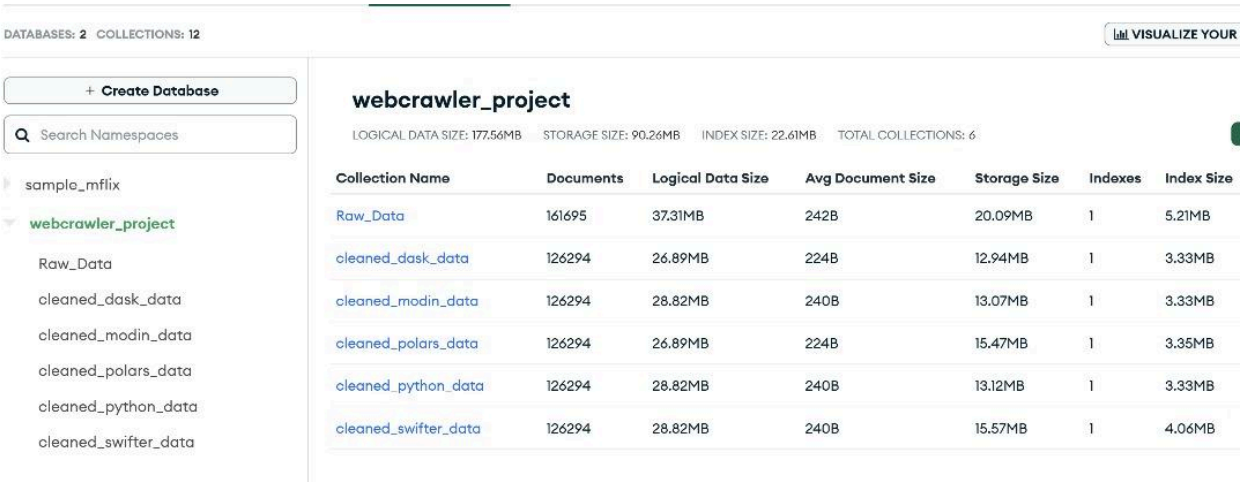


Figure 4.3 Summary of Data Collection in webcrawler_project

5.0 Optimization Techniques

In order to enhance the performance and efficiency of data processing, several optimization techniques can be employed. This section outlines the primary optimization methods utilized, including Polars, Modin, Dask, and Swifter. These techniques aim to leverage parallel processing and distributed computing to expedite computations and handle large datasets more effectively.

5.1 Optimization Methods Used

Table 5.1.1 Description of Optimization Methods Used

Library/Methods Used	Description
	• Swifter is a Python library that accelerates the performance of pandas.apply() operations by dynamically selecting the most efficient execution strategy. • It automatically decides whether to use pandas, Dask, or Numba, depending on the size of the dataset and the complexity of the function. • Swifter is especially useful for large-scale data transformations, enabling parallel execution with minimal code changes and maximizing CPU utilization efficiently.

	• Modin is a library that accelerates Pandas by automatically distributing the computation across all of the system's CPUs. • It provides an effortless way to speed up Pandas workflows without requiring significant code changes. • Particularly effective for large datasets, as it parallelizes Pandas operations, allowing for faster data manipulation and analysis.
	• Dask is a flexible parallel computing library for Python that scales Pandas and NumPy workflows. It enables parallel computing on single machines or distributed clusters. • Dask excels at handling larger-than-memory datasets by breaking them into smaller chunks and processing them in parallel. • It provides high-level abstractions for parallel arrays, dataframes, and machine learning, making it easier to parallelize complex computations.

	• Polars is an open-source library for data manipulation, known for being one of the fastest data processing solutions on a single machine. • Polars uses lazy evaluation, which allows it to build a query plan first then optimize the entire pipeline before execution. • Its execution engine uses multithreading under the hood to parallelize operations.

5.2 Code Overview of Techniques Applied

5.2.1 Swifter

The performance improvements observed in this part of the data cleaning pipeline stem from Swifter's ability to automatically optimize .apply() operations using parallel processing or vectorized execution, depending on the workload size and complexity.

1. *import swifter:* This line introduces the Swifter library, a lightweight yet powerful tool designed to speed up Pandas' .apply() operations without requiring major code changes. Swifter intelligently determines the best execution strategy which could be vectorized, Dask-based, or multiprocessing, to accelerate row-wise operations. The import is seamless and integrates directly with the standard Pandas workflow.
2. *df['Date'] = df['Date'].swifter.apply(convert_date):* This is the core performance enhancement line. In the non-optimized version, the .apply() function is executed sequentially, which can be slow for large datasets. By replacing .apply() with .swifter.apply(), the convert_date function is applied in parallel across multiple cores, improving speed without changing the function logic.
3. *swifter_start_time = time.time()*

   ...

   swifter_execution_time = swifter_end_time - swifter_start_time

   swifter_throughput = len(df) / swifter_execution_time : To complement the optimization, detailed performance tracking was added to benchmark Swifter's efficiency. By isolating timing and computing throughput (i.e., rows processed per second), we were able to quantify the performance gains relative to the unoptimized Pandas version. This provided valuable insights into the resource savings and speedups achieved through parallelism.

5.2.2 Modin

The performance enhancements observed in this code are largely due to Modin's capability to parallelize operations, a feature that is effectively enabled by its integration with the Ray execution engine.

1. Instead of the conventional Pandas library, Modin is imported as 'pd'. Modin is designed to be a drop-in replacement for Pandas, offering an identical API but with significantly improved performance, especially for larger datasets. This performance gain is achieved through Modin's ability to parallelize Pandas operations across multiple CPU cores.

```
import modin.pandas as pd
```

2. This line explicitly instructs Modin to utilize the Ray framework as its execution engine. Ray is a powerful distributed computing framework that facilitates the parallel execution of tasks. By setting the 'MODIN_ENGINE' environment variable to 'ray', we enable Modin to leverage Ray's capabilities to distribute both data and computations. This allows Modin to efficiently utilize available system resources, leading to substantial speedups in data processing.

```
import os
os.environ["MODIN_ENGINE"] = "ray"
```

5.2.3 Dask

Dask's ability to handle large-scale data processing allows for optimizations through parallelism which makes it suitable for datasets that exceed memory limits or require high throughput processing.

1. Imports the Dask library, aliasing it to dd for easier use. The dataset is converted into a Dask dataFrame, which splits the data into multiple smaller partitions and processes them in parallel.

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

2. The number of partitions is dynamically set based on the available CPU cores, ensuring optimal parallelism. This enables each partition to be processed concurrently, significantly reducing execution time.

```
# Optimal partition count
npartitions = multiprocessing.cpu_count()
dask_df = dd.from_pandas(df, npartitions=npartitions)
```

3. "map_partitions" applied the cleaning logic independently to each partition, which preserves the lazy evaluation model and ensures that operations like string parsing and datetime conversion are only performed when compute() is explicitly called.

```
# Apply cleaning in partitions
dask_df = dask_df.map_partitions(clean_partition)

# Step 7: Drop duplicates by URL
dask_df = dask_df.drop_duplicates(subset='URL')

# Final compute
cleaned_dask_df = dask_df.compute()
```

5.2.4 Polars

Leverages lazy evaluation through its LazyFrame. Instead of executing operations immediately, Polars builds an optimized query plan. The actual computation only happens when you call .collect() on the LazyFrame. This allows Polars to analyze the entire sequence of operations and potentially optimize them for speed and memory efficiency.

1. Imports the Polars library, aliasing it to pl for easier use.

```
import polars as pl
```

2. This Polars code segment focuses on efficiently cleaning and preparing a DataFrame using lazy evaluation. It begins by converting the input DataFrame into a LazyFrame, which delays the execution of operations to allow Polars to optimize the entire query plan. The subsequent chained operations define this plan: rows with null values are removed, and rows with the string "no date" in the "Date" column are filtered out. The "Category" column is standardized by capitalizing the first letter of each word, and the "Date" column undergoes transformation to handle Malay month names and is parsed

into a datetime format. Finally, rows with parsing failures in the "Date" column are removed, and the data is deduplicated based on the "URL" column. The collect() function then executes this optimized query plan, producing the cleaned DataFrame.

```
# Convert to LazyFrame for optimization
lazy_df = df_polars.lazy()

# Clean & prepare columns
lazy_df = (
    lazy_df
    .drop_nulls()
    .filter(~pl.col("Date").str.to_lowercase().str.contains("no date"))
    .with_columns([
        pl.col("Category").str.to_titlecase().alias("Category"),
        pl.col("Date").map_elements(replace_malay_months, return_dtype=pl.Utf8)
    ])
    .with_columns([
        pl.col("Date")
        .str.replace_all(r"(am|pm)", "")
        .str.replace_all(",", "")
        .str.strip_chars()
        .str.strptime(pl.Datetime, format="%d %m %Y %I:%M %p", strict=False)
        .alias("Date")
    ])
    .drop_nulls(subset=["Date"])  # Remove failed datetime parsing rows
    .unique(subset=["URL"])       # Deduplicate by URL after date validation
)

# Execute optimized plan
df_cleaned = lazy_df.collect()
```

6.0 Performance Evaluation

This section presents the performance comparison of the data processing before and after optimization. The performance metrics measured include execution time, CPU usage, memory usage, and throughput across different libraries and optimization methods. There are five libraries used to do the cleaning process which are Pure Python (Pandas), Polars, Modin, Dask, and Swifter. Pure Python served as the baseline with no optimization, while the other four libraries are optimized libraries.

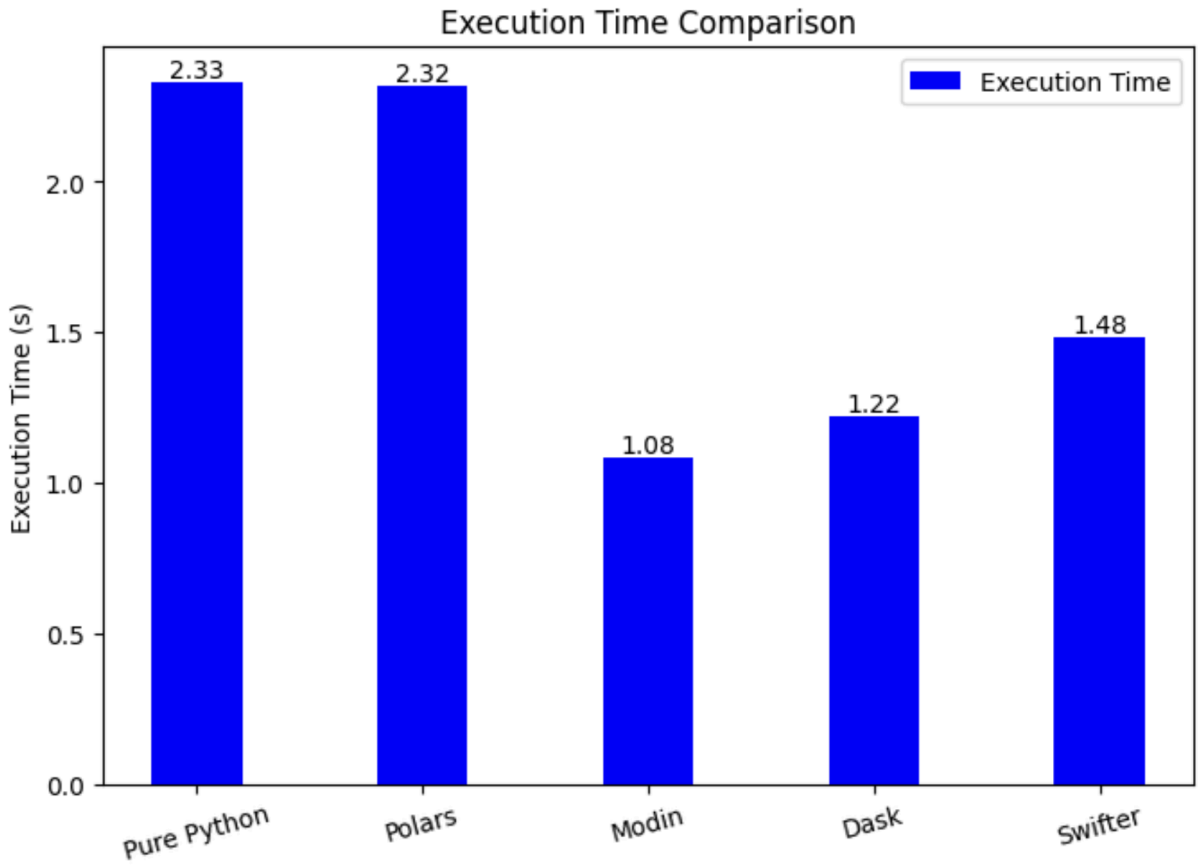


Figure 6.1 Comparison of Execution Time between Libraries

Based on Figure 6.1, the running times of different libraries were compared: Pandas (without optimization), Polars, Modin, Dask, and Swifter. Pandas with no optimization had the longest running time, and Modin had the shortest, which indicates that Modin is the most efficient as the optimized library in this case. Polars presents a minimal improvement with only 0.01 seconds

faster than Pandas. However, Modin, Dask, and Swifter show significant decreases in running time most likely due to the parallelized and distributed data processing. The relatively poor performance of Polars suggests that it may not be best suited for dealing with smaller data sets in the range of 100,000 to 200,000 rows, likely due to the overhead of its distributed computing environment, which is typically optimized for dealing with much larger data sets.

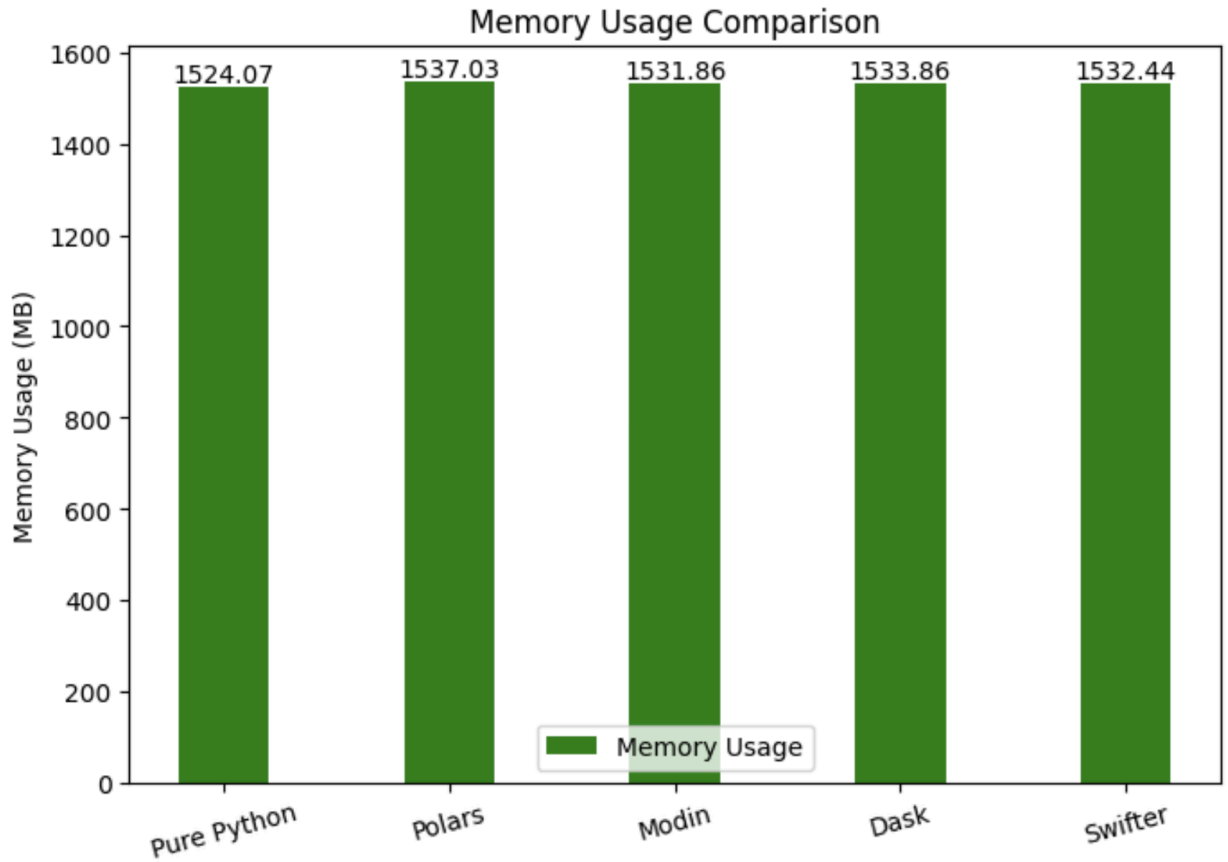


Figure 6.2 Comparison of Memory Usage between Libraries

Based on Figure 6.2, the bar chart presents consistent consumption across all libraries. The memory usage values range from approximately 1524 MB to 1537 MB. This result indicates that despite the optimized library significantly improving speed and efficiency, memory usage remains fairly constant, which may be attributed to the uniform size and structure of the dataset.

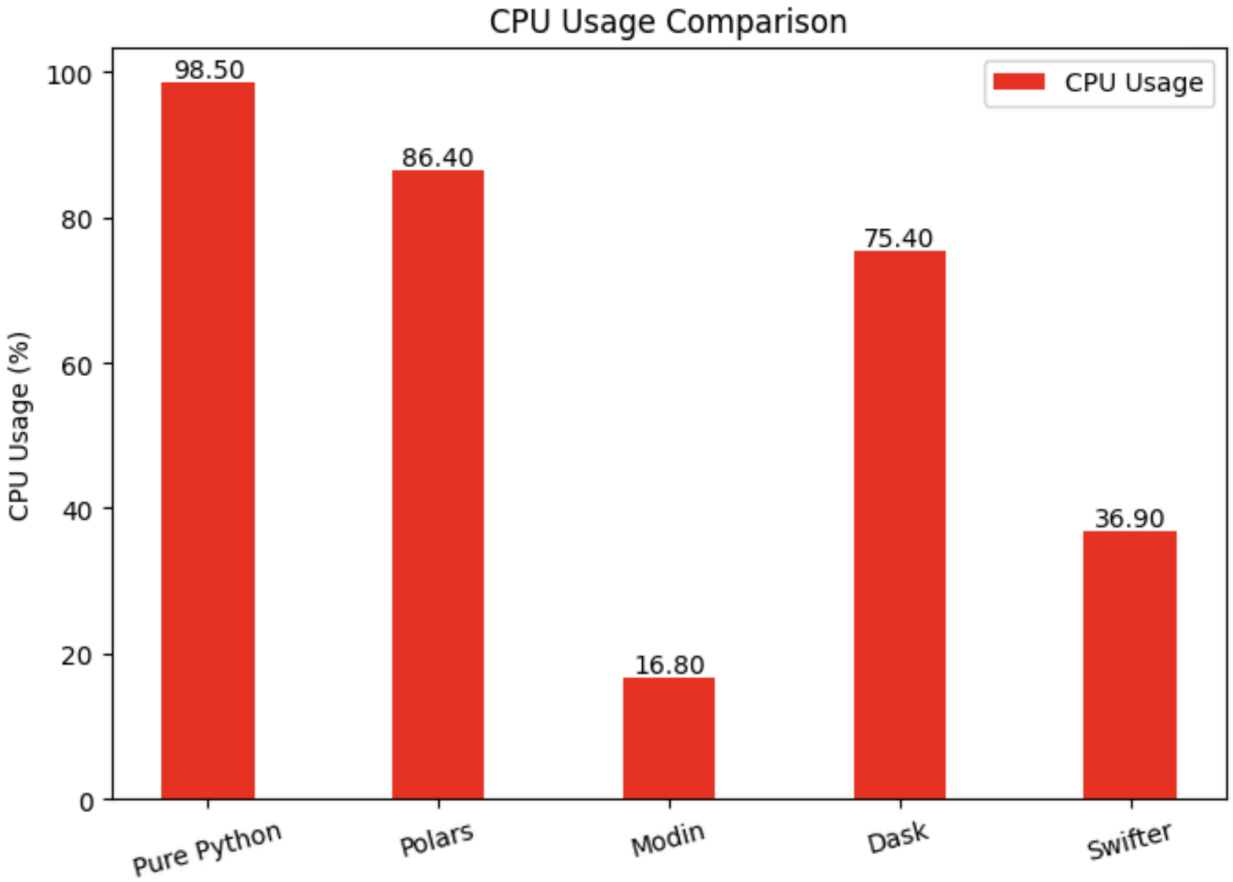


Figure 6.3 Comparison of CPUUsage between Libraries

Figure 6.3 illustrates the CPU usage comparison was made among different libraries. Pure Python (Pandas) showed the highest CPU usage, representing that without optimization, it is inefficient for big data processing. However, Swifter showed high CPU usage (36.9%) but still slower than Modin (16.8%) and Dask (74.4%). Additionally, Polars also showed high CPU usage with high execution time which can appear it is less efficient for local or small-scale setups. Modin's low CPU usage due to the efficient distributed task across cores using Ray. Whereas the slightly more CPU consumption for Dask is most likely due to task scheduling across its partitions while parallelized.

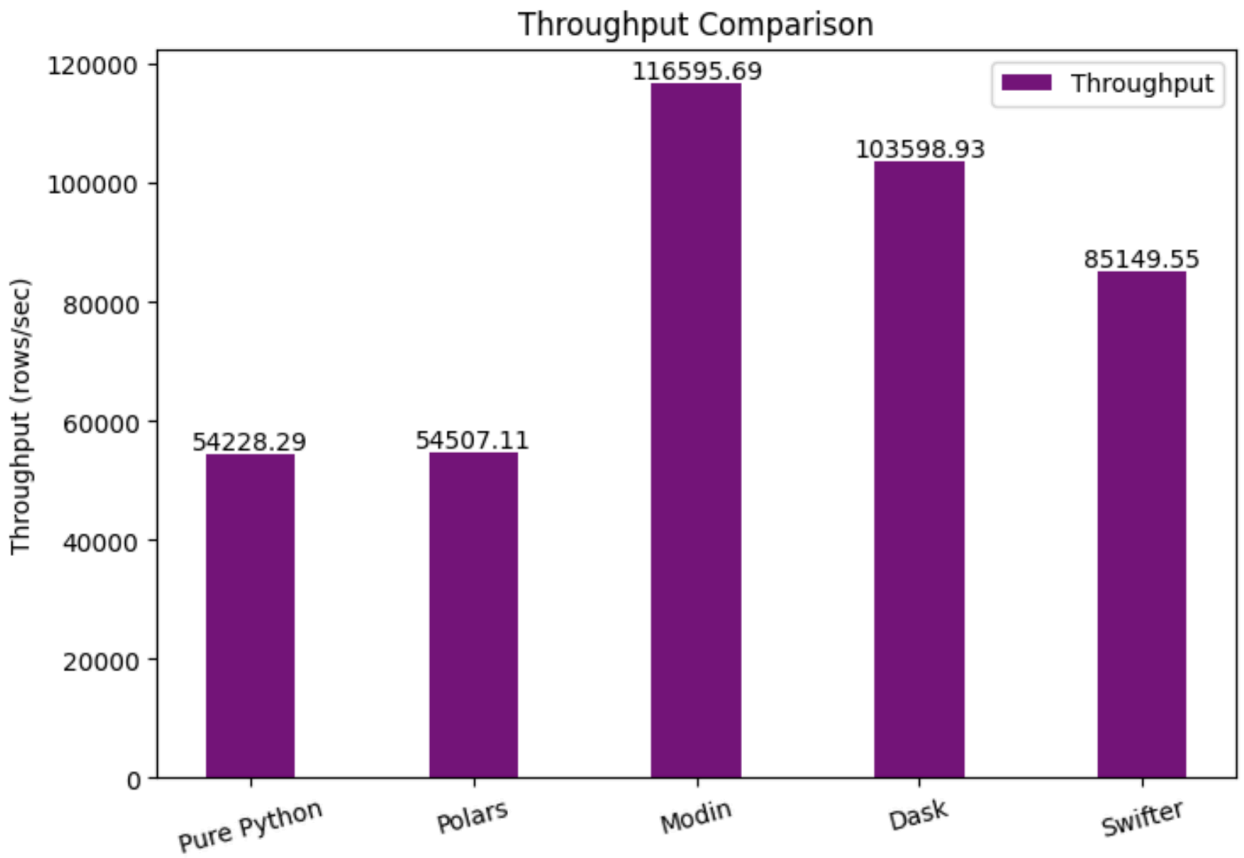


Figure 6.4 Comparison of Throughput between Libraries

Based on Figure 6.4, it demonstrates the comparison of throughput of different libraries. As illustrated by the chart, Modin achieved the highest throughput, followed by Dask and Swifter. These findings prove that parallel computing libraries can increase the rate of data processing compared to single-threaded execution. Pure Python (Pandas) and Polars showed the lowest throughput, around 54,000 rows per second, aligning with their slower overall execution time. Therefore, Polars as an optimized library used in this case appear to be the most inefficient in overall performance.

```
Comparison Table (Performance Metrics by Library):

    Library  Final Row Count  Execution Time (s)  CPU Usage (%)  Memory Usage (MB)  Throughput (rows/sec)
Pure Python           126294            2.328932           98.5        1524.066406           54228.288957
     Polars           126294            2.317019           86.4        1537.031250           54507.106233
      Modin           126294            1.083179           16.8        1531.855469          116595.687643
       Dask           126294            1.219067           75.4        1533.863281          103598.932118
    Swifter           126294            1.483202           36.9        1532.441406           85149.548984
```

Figure 6.5 Comparison Table of Performance Metrics

Figure 6.5 illustrates a summary of performance metrics across five libraries in a table for easier comparison. The optimization techniques clearly demonstrated a significant improvement in execution time and processing efficiency. As shown in the performance metrics table, Modin consistently outperformed other methods in both speed and throughput, making it appear as the best approach for efficient data cleaning with minimal code changes. Dask also showed strong performance, but it will be more useful when scaling to larger datasets. Swifter offered a lightweight optimization path with reasonable performance improvements, especially when code simplicity is prioritized. On the other hand, Pure Python and Polars demonstrated less compelling results for the dataset size tested, with Polars likely constrained by overhead rather than computational inefficiency.

7.0 Challenges and Limitations

There are several challenges that were encountered throughout the development of this project, which are detailed below:

1. Target Website Changes: Initially, the project aimed to scrape the CompAsia website, then The Star for web crawlers. However, during the early data collection phase, we realized that CompAisa had limited crawlable content that consisted mainly of e-commerce products with very little structural variation. The Star had access limitations and pagination issues that significantly limited the scale and speed of data extraction. Due to these challenges, we shifted our focus to Utusan Malaysia, which offered a nearly arranged archive, systematic category tagging, and a sufficient amount of data to support a large-scale web crawling and analysis project.

2. Limitations of BeautifulSoup for dynamic content: Crawler 2 was designed to use only the BeautifulSoup library, but we discovered that BeautifulSoup alone is insufficient to handle dynamic websites. To address this issue, we integrated Selenium, which can render dynamic content.

3. Complexity in integrating the Scrapy framework: In evaluating web crawling frameworks, we considered using Scrapy, known as a powerful web scraping framework. However, we found that Scrapy is too complex in our case and was not fully compatible with Google Colab due to many dependencies required for installation before implementing the crawler. As a result, we opted to use a more modular approach like Requests, BeautifulSoup, Selenium, and Selectolax.

8.0 Conclusion and Future Work

8.1 Summary of Findings

Throughout this project, our team set out to build an efficient and scalable solution to extract and process large-scale data from the Utusan Malaysia website. With a clear target of collecting over 100,000 news records, we combined both traditional and modern tools, including Selenium and Playwright for crawling, and BeautifulSoup for parsing. For data cleaning and processing, we incorporated performance-oriented libraries like Modin, Dask, PySpark, and Swifter.

One of our main achievements was optimizing the crawler to systematically extract articles across multiple sections under the "Gaya" category, such as *agama*, *fesyen*, and *kesihatan*. With multithreading, we significantly cut down crawling time, and through intelligent page navigation and parsing strategies, we ensured consistency in the collected records.

On the processing side, we benchmarked different HPC techniques and observed that:

- Modin and Dask helped speed up typical Pandas operations without needing much code change.

- Swifter gave moderate speedups for row-wise operations, useful during data cleaning.

- Polars provided significant performance improvements, particularly for columnar operations and when optimizing query-like operations on DataFrames.

Overall, the project successfully demonstrated how high-performance computing tools can be integrated into real-world data crawling tasks, providing both scalability and efficiency.

8.2 Future Work

While we met our core objectives, there are still several areas where improvements and extensions could make the system even more robust and future-ready:

1. Scalability with Distributed Crawlers

   Currently, the crawler runs on a single machine with multithreading. In the future, setting up distributed crawling (e.g., with Scrapy Cluster or Kafka pipelines) could help handle even larger websites or multiple news sources in parallel.

2. Improved Fault Tolerance

   Some sections occasionally failed due to JavaScript-heavy pages or connection timeouts. Building in smarter retry logic and better error logging would reduce data loss and ease debugging.

3. Incremental Crawling

   At present, our crawler collects all articles in a single run. Adding support for incremental updates — such as crawling only new or updated articles daily — would reduce redundancy and keep the dataset fresh.

4. Cloud-Based Processing

   While we tested everything locally, deploying PySpark or Dask clusters on the cloud (e.g., AWS, GCP) would offer better scalability and reduce runtime for massive datasets.

5. Post-Crawling Analysis

   With such a rich dataset, there's potential for further analysis — such as identifying trends in Malaysian lifestyle topics, sentiment shifts over time, or topic clustering using NLP techniques.

This project has given us valuable hands-on experience in dealing with both large-scale crawling and high-performance processing and sets the groundwork for building even more advanced data pipelines in the future.

9.0 References

Modin:

1. https://docs.ray.io/en/latest/ray-more-libs/modin/index.html
2. https://modin.readthedocs.io/en/latest/index.html

Dask:

1. https://www.kdnuggets.com/introduction-dask-python-data-scientist-power-tool#:~:text=Dask%20splits%20these%20large%20datasets,process%20of%20handling%20big%20data.

Polars:

1. https://www.linkedin.com/pulse/high-performance-data-analysis-polars-comprehensive-r0rdf/
2. https://docs.pola.rs/#key-features

Swifter:

1. https://www.imranabdullah.com/2022-03-30/faster-pandas-operation-using-swifter

10.0 Appendices

Sample Code Snippets

Web Scraping

Section	Screenshot
Nasional_Scrape(BeautifulSoup_Playwright).ipynb	

```
!pip install -q playwright
!playwright install

import asyncio
from playwright.async_api import async_playwright
from bs4 import BeautifulSoup
import csv

# Settings
MAX_PAGES = 900  # How many pages to scrape PER tab
TABS = ['jenayah', 'mahkamah', 'tragedi']  # Add or remove tabs as needed
OUTPUT_CSV = 'utusan_nasional_combined.csv'

async def run():
    with open(OUTPUT_CSV, 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        # CSV column header
        writer.writerow(['Title', 'Date', 'Category', 'Tab', 'URL'])

        async with async_playwright() as p:
            browser = await p.chromium.launch(headless=True)
            page = await browser.new_page()

            for tab in TABS:
                print(f"\n📁 Starting tab: {tab}")

                for page_num in range(1, MAX_PAGES + 1):
                    url = f'https://www.utusan.com.my/category/nasional/{tab}/page/{page_num}/'
                    print(f"📄 Scraping {tab.capitalize()} - Page {page_num}")

                    try:
                        await page.goto(url, timeout=500_000)
                        await page.wait_for_timeout(8000)  # Wait for full loading
                        html = await page.content()
                        soup = BeautifulSoup(html, 'html.parser')
                        articles = soup.select('article.jeg_post')
```

Ekonomi_Ringgit_LuarNegara.ipynb	

```python
def scrape_page(url):
    for attempt in range(3):
        try:
            print(f"Loading: {url} (Attempt {attempt + 1})")
            response = requests.get(url, timeout=MAX_PAGE_TIMEOUT)
            if response.status_code == 200:
                return response.text
            print(f"⚠️ Error: {response.status_code}")
        except Exception as e:
            print(f"⚠️ Exception: {e}")
        time.sleep(RETRY_WAIT)
    print(f"❌ Failed to load {url}")
    return None

def scrape_category(relative_path, category_name, writer, max_records):
    total_scraped = 0
    page_num = 1

    while total_scraped < max_records:
        # Handle pagination logic
        url = f"{BASE_URL}{relative_path}" if category_name in NO_PAGINATION else f"{BASE_URL}{relative_path}page/{page_num}/"
        html = scrape_page(url)
        if not html:
            break

        soup = BeautifulSoup(html, 'html.parser')
        articles = soup.find_all('article', class_='jeg_post')
        if not articles:
            break

        for article in articles:
            if total_scraped >= max_records:
                break
            title_tag = article.find('h3', class_='jeg_post_title')
            title = title_tag.get_text(strip=True) if title_tag else 'No Title'
            link_tag = title_tag.find('a') if title_tag else None
            full_url = link_tag['href'] if link_tag and 'href' in link_tag.attrs else 'No URL'

            date_tag = article.find('div', class_='jeg_meta_date')
            date = date_tag.get_text(strip=True) if date_tag else 'No Date'

            # Use provided dictionary name as the category
            cat = category_name.capitalize()

            writer.writerow([title, full_url, date, cat])
            total_scraped += 1


        print(f"✅ {category_name.capitalize()} - Page {page_num} done. Total scraped: {total_scraped}")
```

Politik_Sukan_Rencana_Pancaindera_Selectolax.ipynb	

```
###Use Selectolax library

import requests
import pandas as pd
import time
from selectolax.parser import HTMLParser

# Subcategories under 'Pancaindera' and 'Politik' and 'Sukan' and 'Rencana'
subcategories = {
    'cover': 'https://www.utusan.com.my/category/pancaindera/cover/',
    'khabar-lagenda': 'https://www.utusan.com.my/category/pancaindera/khabar-lagenda/',
    'rondivu': 'https://www.utusan.com.my/category/pancaindera/rondivu/',
    'hangat-seminggu': 'https://www.utusan.com.my/category/pancaindera/hangat-seminggu/',
    'jawablah': 'https://www.utusan.com.my/category/pancaindera/jawablah/',
    'jejak-ulama': 'https://www.utusan.com.my/category/pancaindera/jejak-ulama/',
    'sastera': 'https://www.utusan.com.my/category/gaya/sastera/',
    'seram': 'https://www.utusan.com.my/category/pancaindera/seram/',
    'politik': 'https://www.utusan.com.my/category/nasional/politik/',
     'bola-sepak': 'https://www.utusan.com.my/category/sukan/bola-sepak/',
    'badminton': 'https://www.utusan.com.my/category/sukan/badminton/',
    'e-sukan': 'https://www.utusan.com.my/category/sukan/e-sukan/',
    'lumba-basikal': 'https://www.utusan.com.my/category/sukan/basikal/',
    'lumba-kereta': 'https://www.utusan.com.my/category/sukan/lumba-kereta/',
    'sepak-takraw': 'https://www.utusan.com.my/category/sukan/sepak-takraw/',
    'surat-pembaca': 'https://www.utusan.com.my/category/rencana/surat-pembaca/',
    'wawancara': 'https://www.utusan.com.my/category/rencana/wawancara/',
}

# Function to scrape a single page using selectolax
def scrape_page(url, category_name):
    response = requests.get(url)
    response.raise_for_status()
    tree = HTMLParser(response.text)

    articles = tree.css('article.jeg_post')
    page_data = []

    for article in articles:
        title_node = article.css_first('h3.jeg_post_title a')
        title = title_node.text(strip=True) if title_node else None
        link = title_node.attrs.get('href') if title_node else None

        img_node = article.css_first('img')
        img_url = img_node.attrs.get('src') if img_node else None
```

	`` category_node = article.css_first('div.jeg_post_category') category = category_node.text(strip=True) if category_node else None date_node = article.css_first('div.jeg_meta_date') date = date_node.text(strip=True) if date_node else None page_data.append({ 'Title': title, 'Link': link, 'Image_URL': img_url, 'Category': category, 'Subcategory': category_name, 'DateTime': date }) return page_data # Scraping logic all_data = [] for sub_name, sub_url in subcategories.items(): print(f"--- Scraping subcategory: {sub_name.upper()} ---") page = 1 while True: page_url = sub_url if page == 1 else f"{sub_url}page/{page}/" print(f"Scraping page {page}: {page_url}") try: page_data = scrape_page(page_url, sub_name) if not page_data: print(f"No more articles found on page {page}. Stopping.") break all_data.extend(page_data) page += 1 time.sleep(1) except Exception as e: print(f"Failed to scrape page {page} of {sub_name}: {e}") break # Save results to CSV df = pd.DataFrame(all_data) df.to_csv('utusan_pancaindera_all_pages.csv', index=False) print("✅ Scraping completed! Saved to 'utusan_pancaindera_all_pages.csv'.") ```
Gaya_Komuniti_Selenium.ipynb	``` from selenium import webdriver from selenium.webdriver.chrome.service import Service from selenium.webdriver.chrome.options import Options from selenium.webdriver.common.by import By import csv import time import random import os # Setup Selenium ChromeDriver (headless mode) def setup_driver(): options = Options() options.add_argument('--headless') # Comment this out to see browser window options.add_argument('--disable-gpu') options.add_argument('--no-sandbox') service = Service() driver = webdriver.Chrome(service=service, options=options) return driver # Scrape a single subcategory def scrape_subcategory(driver, category, subcategory, writer, max_pages=1400, max_records=50000): base_url = f'https://www.utusan.com.my/category/{category}/{subcategory}/page/' page_num = 1 total_scraped = 0 ```

```
    while total_scraped < max_records and page_num <= max_pages:
        url = f"{base_url}{page_num}/"
        print(f"🔄 Loading {category}/{subcategory} - Page {page_num}: {url}")
        try:
            driver.get(url)
            time.sleep(random.uniform(1, 3))  # Allow page to load

            articles = driver.find_elements(By.CLASS_NAME, 'jeg_post')
            if not articles:
                print(f"⚠️ No articles found on page {page_num} for {subcategory}. Stopping.")
                break

            for article in articles:
                if total_scraped >= max_records:
                    break

                try:
                    title_elem = article.find_element(By.CLASS_NAME, 'jeg_post_title')
                    title = title_elem.text.strip()
                    link = title_elem.find_element(By.TAG_NAME, 'a').get_attribute('href')
                except:
                    title, link = 'No Title', 'No URL'

                try:
                    date = article.find_element(By.CLASS_NAME, 'jeg_meta_date').text.strip()
                except:
                    date = 'No Date'
```

```
                writer.writerow([title, link, date, category, subcategory])
                total_scraped += 1

            print(f"✅ Done page {page_num} of {subcategory}. Total scraped: {total_scraped}")
            page_num += 1

        except Exception as e:
            print(f"❌ Error on {category}/{subcategory} page {page_num}: {e}")
            page_num += 1
            time.sleep(1)

def run_full_scraper():
    driver = setup_driver()

    filename = 'utusan_full_scrape.csv'
    csv_exists = os.path.exists(filename)

    with open(filename, 'a', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        if not csv_exists:
            writer.writerow(['Title', 'URL', 'Date', 'Category', 'Subcategory'])

        # Gaya category and its subcategories
        gaya_subcategories = [
            'agama', 'agro', 'anakku', 'deko', 'fesyen',
            'gajet', 'hiburan', 'keluarga-wanita', 'kesihatan', 'pendidikan'
        ]
```

```
        for subcat in gaya_subcategories:
            scrape_subcategory(driver, 'gaya', subcat, writer)

        # Nasional > Komuniti
        scrape_subcategory(driver, 'nasional', 'komuniti', writer)

    driver.quit()
    print("🎉 Full scraping complete!")

# Run it
run_full_scraper()
```

Export to MongoDB.ipynb - combine all scraped data and load into MongoDB	``` utusan_articles = pd.concat([nasional, erl, srp, gk], ignore_index=True) utusan_articles ``` ``` from pymongo.mongo_client import MongoClient from pymongo.server_api import ServerApi uri = "mongodb+srv://nuraleyshaqurratuaini:Leysha18@cluster0.dsmzjrk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0" # Create a new client and connect to the server client = MongoClient(uri, server_api=ServerApi('1')) # Send a ping to confirm a successful connection try: client.admin.command('ping') print("Pinged your deployment. You successfully connected to MongoDB!") except Exception as e: print(e) ``` ``` db = client["webcrawler_project"] collection = db["Raw_Data"] # Convert to list of dicts (MongoDB format) data_dict = utusan_articles.to_dict("records") # Insert into MongoDB collection.insert_many(data_dict) print("✅ Inserted", len(data_dict), "rows into MongoDB.") ```

Data Processing and Optimization

Section	Screenshot
Cleaning_Optimization_Comparison.ipynb	

```python
def clean_with_modin(df):

    start_time = time.time()
    start_memory = psutil.Process().memory_info().rss / (1024 * 1024)

    # Initial cleaning
    modin = df[['Title', 'Category', 'Date', 'URL']].dropna().copy()
    modin = modin[~modin['Date'].astype(str).str.strip().str.lower().eq('no date')]
    modin['Category'] = modin['Category'].astype(str).str.capitalize()

    month_map = {
        'januari': '01', 'februari': '02', 'mac': '03', 'april': '04',
        'mei': '05', 'jun': '06', 'julai': '07', 'ogos': '08',
        'september': '09', 'oktober': '10', 'november': '11', 'disember': '12'
    }

    def convert_date(s):
        try:
            m = re.match(r'(\d{1,2}) (\w+) (\d{4}), (\d{1,2}):(\d{2}) (\w+)', str(s).lower())
            if m:
                day, month, year, hour, minute, ampm = m.groups()
                if ampm == 'pm' and int(hour) != 12:
                    hour = str(int(hour) + 12)
                elif ampm == 'am' and int(hour) == 12:
                    hour = '00'
                return f"{year}-{month_map.get(month)}-{day.zfill(2)} {hour.zfill(2)}:{minute}:00"
        except:
            return np.nan
        return np.nan

    modin['Date'] = modin['Date'].apply(convert_date)
    modin.dropna(subset=['Date'], inplace=True)

    modin.drop_duplicates(subset='URL', keep='first', inplace=True)

    # Track performance
    modin_metrics = track_performance("Modin", start_time, start_memory, modin)

    return modin, modin_metrics

modin_cleaned, modin_metrics = clean_with_modin(df)
```

```python
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
import os

# Connect to MongoDB
client = MongoClient("mongodb+srv://nuraleyshaqurratuaini:Leysha18@cluster0.dsmzjrk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client["webcrawler_project"]
# collection = db["cleaned_data_db"]

# Output folder
output_dir = "cleaned_exports"
os.makedirs(output_dir, exist_ok=True)

# List of (DataFrame, CSV name, MongoDB collection)
exports = [
    (pure_python_cleaned, "cleaned_python.csv", "cleaned_python_data"),
    (cleaned_polars_df.to_pandas(), "cleaned_polars.csv", "cleaned_polars_data"),
    (modin_cleaned, "cleaned_modin.csv", "cleaned_modin_data"),
    (cleaned_dask_df, "cleaned_dask.csv", "cleaned_dask_data"),
    (cleaned_swifter_df, "cleaned_swifter.csv", "cleaned_swifter_data")
]

for df, filename, collection_name in exports:
    # Save to CSV
    csv_path = os.path.join(output_dir, filename)
    df.to_csv(csv_path, index=False)
    print(f"✅ Saved: {filename}")

    # Insert into MongoDB
    mongo_collection = db[collection_name]
    mongo_collection.delete_many({})  # Clear existing records
    mongo_collection.insert_many(df.to_dict(orient="records"))
    print(f"📤 Uploaded to MongoDB collection: {collection_name}")
```

```
# Labels for bars
labels = list(stats.keys())

# Extract each metric across all methods
execution_times = [stats[lib]["Execution Time (s)"] for lib in labels]
cpu_usages = [stats[lib]["CPU Usage (%)"] for lib in labels]
memory_usages = [stats[lib]["Memory Usage (MB)"] for lib in labels]
throughputs = [stats[lib]["Throughput (rows/sec)"] for lib in labels]

x = np.arange(len(labels))
width = 0.4

fig, axs = plt.subplots(2, 2, figsize=(14, 10))

# Execution Time
bars1 = axs[0, 0].bar(x, execution_times, width, color='blue')
axs[0, 0].set_title('Execution Time Comparison')
axs[0, 0].set_ylabel('Execution Time (s)')
axs[0, 0].set_xticks(x)
axs[0, 0].set_xticklabels(labels, rotation=15)
axs[0, 0].legend(["Execution Time"])
axs[0, 0].bar_label(bars1, fmt='%.2f')
```

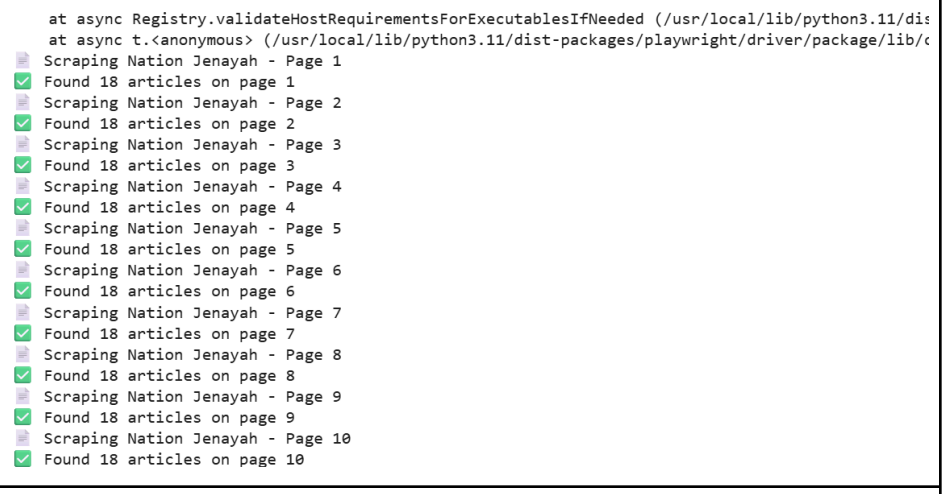
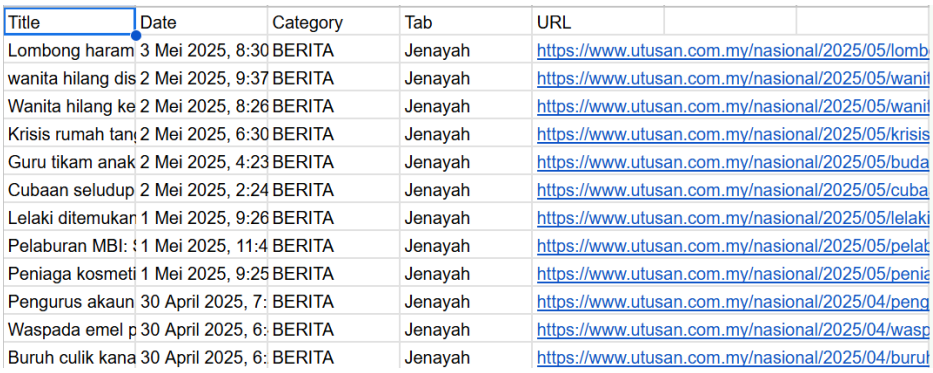
```
# Stats for each library (ensure these variables are already defined)
stats = {
    "Pure Python": pure_python_metrics,
    "Polars": polars_metrics,
    "Modin": modin_metrics,
    "Dask": dask_metrics,
    "Swifter": swifter_metrics
}

# Convert to DataFrame for tabular display
df = pd.DataFrame.from_dict(stats, orient='index')
df.index.name = 'Library'
df = df.reset_index()

# Display the table
print("\nComparison Table (Performance Metrics by Library):\n")
print(df.to_string(index=False))
```

Screenshots of Output

Web Scraping Output

Section	Screenshot
Nasional.ipynb	
Nasional.csv	
Ekonomi_Ringgit_LuarNegara.ipynb	

Ekonomi_Ringgit_LuarNegara.csv

Title	URL	Date	Category
Batiq jadikan S2	Daniel Hakim tus	5 Mei 2025, 7:21	Hartanah
Agensi perumah	https://www.utus	28 April 2025, 9:	Hartanah
Bangi Fresco tav	https://www.utus	28 April 2025, 8:	Hartanah
SouthPlace Resi	https://www.utus	21 April 2025, 7:	Hartanah
Taman Lang Am	https://www.utus	7 April 2025, 7:2	Hartanah
Matrix Concepts	https://www.utus	24 Mac 2025, 7:	Hartanah
Freesia kediama	https://www.utus	17 Mac 2025, 7:	Hartanah
Muhibah Aset ba	https://www.utus	13 Mac 2025, 10	Hartanah
Stellaris @ Rian	https://www.utus	10 Mac 2025, 7:	Hartanah
The Clove capai	https://www.utus	3 Mac 2025, 6:5	Hartanah
ALAIA Titiwangs	https://www.utus	24 Februari 202	Hartanah
Iringan Bayu per	https://www.utus	17 Februari 202	Hartanah
ViO Banj'ran, ba	https://www.utus	10 Februari 202	Hartanah
The Ria, apartm	https://www.utus	3 Februari 2025,	Hartanah
Fraser Heights n	https://www.utus	27 Januari 2025,	Hartanah

Gaya_Komuniti_Selenium.ipynb

```
Loading nasional/komuniti - Page 1353: https://www.utusan.com.my/category/nasional/komuniti/page/1353/
Done page 1353 of komuniti. Total scraped: 24354
Loading nasional/komuniti - Page 1354: https://www.utusan.com.my/category/nasional/komuniti/page/1354/
Done page 1354 of komuniti. Total scraped: 24372
Loading nasional/komuniti - Page 1355: https://www.utusan.com.my/category/nasional/komuniti/page/1355/
Done page 1355 of komuniti. Total scraped: 24390
Loading nasional/komuniti - Page 1356: https://www.utusan.com.my/category/nasional/komuniti/page/1356/
Done page 1356 of komuniti. Total scraped: 24408
Loading nasional/komuniti - Page 1357: https://www.utusan.com.my/category/nasional/komuniti/page/1357/
Done page 1357 of komuniti. Total scraped: 24426
Loading nasional/komuniti - Page 1358: https://www.utusan.com.my/category/nasional/komuniti/page/1358/
Done page 1358 of komuniti. Total scraped: 24444
Loading nasional/komuniti - Page 1359: https://www.utusan.com.my/category/nasional/komuniti/page/1359/
Done page 1359 of komuniti. Total scraped: 24462
Loading nasional/komuniti - Page 1360: https://www.utusan.com.my/category/nasional/komuniti/page/1360/
Done page 1360 of komuniti. Total scraped: 24480
Loading nasional/komuniti - Page 1361: https://www.utusan.com.my/category/nasional/komuniti/page/1361/
Done page 1361 of komuniti. Total scraped: 24498
Loading nasional/komuniti - Page 1362: https://www.utusan.com.my/category/nasional/komuniti/page/1362/
Done page 1362 of komuniti. Total scraped: 24516
Loading nasional/komuniti - Page 1363: https://www.utusan.com.my/category/nasional/komuniti/page/1363/
Done page 1363 of komuniti. Total scraped: 24534
Loading nasional/komuniti - Page 1364: https://www.utusan.com.my/category/nasional/komuniti/page/1364/
Done page 1364 of komuniti. Total scraped: 24552
Loading nasional/komuniti - Page 1365: https://www.utusan.com.my/category/nasional/komuniti/page/1365/
Done page 1365 of komuniti. Total scraped: 24570
Loading nasional/komuniti - Page 1366: https://www.utusan.com.my/category/nasional/komuniti/page/1366/
Done page 1366 of komuniti. Total scraped: 24571
Loading nasional/komuniti - Page 1367: https://www.utusan.com.my/category/nasional/komuniti/page/1367/
No articles found on page 1367 for komuniti. Stopping.
Full scraping complete!
```

Gaya_Komuniti_Selenium.csv

	Title	URL	Date	Category	Subcategory
2	Kerja! Jangan mencuri!	https://www.utusan.com.my/gaya/2025/05/kerja-jangan-mencuri/	2 Mei 2025, 8:00 am	gaya	agama
3	Putus tangan pertahan bendera	https://www.utusan.com.my/gaya/2025/04/putus-tangan-pertahan-bende	18 April 2025, 08:00	gaya	agama
4	Penduduk Melayu Katanning, Australia Barat perlu guru agama	https://www.utusan.com.my/nasional/2025/04/penduduk-melayu-katanni	15 April 2025, 15:02	gaya	agama
5	Gabung dua program dekatkan masyarakat dengan masjid	https://www.utusan.com.my/nasional/2025/04/gabung-dua-program-deka	12 April 2025, 08:45	gaya	agama
6	Seputar mengenai al-Quran	https://www.utusan.com.my/gaya/2025/04/seputar-mengenai-al-quran/	11 April 2025, 11:00	gaya	agama
7	Musim sakura dengan al-Quran	https://www.utusan.com.my/gaya/2025/04/musim-sakura-dengan-al-qura	4 April 2025, 08:00	gaya	agama
8	Contohi diplomasi, kebijaksanaan kepemimpinan Nabi Sulaiman	https://www.utusan.com.my/gaya/2025/03/contohi-diplomasi-kebijaksana	28 Mac 2025, 10:30 am	gaya	agama
9	Dari langit turun ke jantung	https://www.utusan.com.my/gaya/2025/03/dari-langit-turun-ke-jantung/	21 Mac 2025, 9:00 am	gaya	agama
10	Al-Quran dan hadis qudsi	https://www.utusan.com.my/gaya/2025/03/al-quran-dan-hadis-qudsi/	14 Mac 2025, 10:00 am	gaya	agama
11	Usah jadikan Ramadan 'bulan pesta makan' – Mufti	https://www.utusan.com.my/nasional/2025/03/usah-jadikan-ramadan-bul	13 Mac 2025, 7:30 am	gaya	agama
12	JAIS serah surat kebenaran solat kepada Masjid Al-Falah	https://www.utusan.com.my/nasional/2025/03/jais-serah-surat-kebenara	7 Mac 2025, 11:13 am	gaya	agama
13	Usah perlekeh kedatangan ramai jemaah ke masjid	https://www.utusan.com.my/gaya/agama/2025/03/usah-perlekeh-kedata	3 Mac 2025, 7:30 am	gaya	agama
14	Mengapa logo halal Jakim penting dalam industri makanan?	https://www.utusan.com.my/gaya/2025/02/mengapa-logo-halal-jakim-pe	28 Februari 2025, 10:00 am	gaya	agama
15	Fahaman bertentangan ASWJ di Perak terkawal	https://www.utusan.com.my/berita/2025/02/fahaman-bertentangan-aswj-	21 Februari 2025, 6:28 am	gaya	agama
16	Solat duha tanda syukur pada Allah	https://www.utusan.com.my/gaya/2025/02/solat-duha-tanda-syukur-pada	14 Februari 2025, 8:00 am	gaya	agama
17	Masjid Al-Ikhlas tamatkan penantian lebih 30 tahun penduduk	https://www.utusan.com.my/nasional/2025/02/masjid-al-ikhlas-tamatkan	13 Februari 2025, 8:15 am	gaya	agama
18	16,000 Orang Asli di Pahang peluk Islam	https://www.utusan.com.my/berita/2025/02/16000-orang-asli-di-pahang-i	13 Februari 2025, 7:30 am	gaya	agama
19	Bukan halangan hadiri majlis bukan Islam	https://www.utusan.com.my/nasional/2025/02/bukan-halangan-hadiri-ma	10 Februari 2025, 7:30 am	gaya	agama
20	Berlian suci berbau harum di celahan lumpur	https://www.utusan.com.my/gaya/2025/02/berlian-suci-berbau-harum-di-	7 Februari 2025, 9:00 am	gaya	agama
21	Bayar zakat 33 tahun 'sucikan harta'	https://www.utusan.com.my/berita/2025/02/bayar-zakat-33-tahun-sucika	6 Februari 2025, 8:45 am	gaya	agama

Politik_Sukan_Rencana_Pancaindera_Selectolax.ipynb

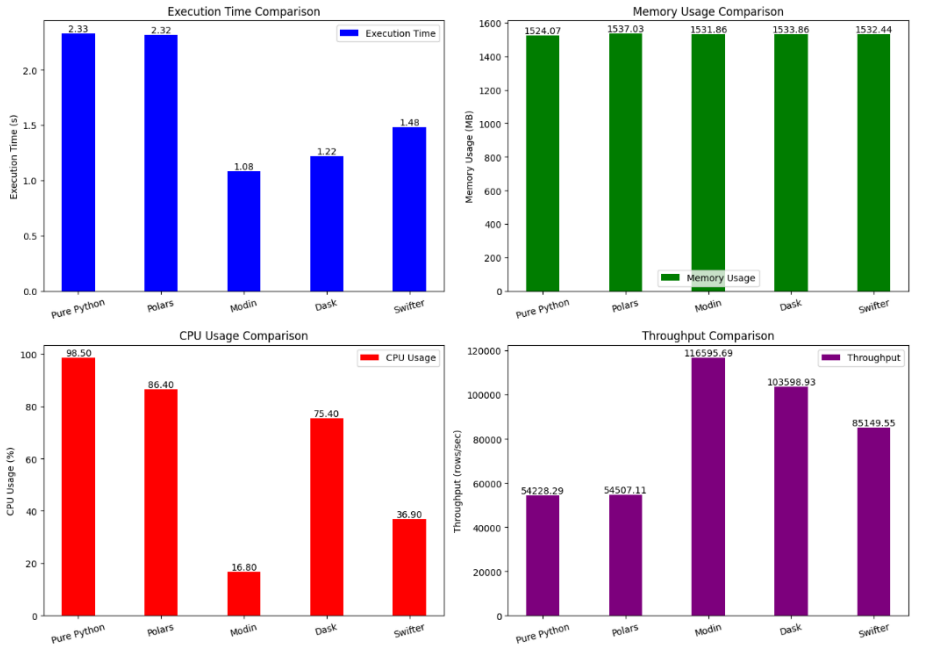
```
--- Scraping subcategory: SASTERA ---
Scraping page 1: https://www.utusan.com.my/category/gaya/sastera/
Scraping page 2: https://www.utusan.com.my/category/gaya/sastera/page/2/
Scraping page 3: https://www.utusan.com.my/category/gaya/sastera/page/3/
Scraping page 4: https://www.utusan.com.my/category/gaya/sastera/page/4/
Scraping page 5: https://www.utusan.com.my/category/gaya/sastera/page/5/
Scraping page 6: https://www.utusan.com.my/category/gaya/sastera/page/6/
Scraping page 7: https://www.utusan.com.my/category/gaya/sastera/page/7/
Scraping page 8: https://www.utusan.com.my/category/gaya/sastera/page/8/
Scraping page 9: https://www.utusan.com.my/category/gaya/sastera/page/9/
Scraping page 10: https://www.utusan.com.my/category/gaya/sastera/page/10/
Scraping page 11: https://www.utusan.com.my/category/gaya/sastera/page/11/
Scraping page 12: https://www.utusan.com.my/category/gaya/sastera/page/12/
Scraping page 13: https://www.utusan.com.my/category/gaya/sastera/page/13/
Scraping page 14: https://www.utusan.com.my/category/gaya/sastera/page/14/
Scraping page 15: https://www.utusan.com.my/category/gaya/sastera/page/15/
Scraping page 16: https://www.utusan.com.my/category/gaya/sastera/page/16/
Scraping page 17: https://www.utusan.com.my/category/gaya/sastera/page/17/
Scraping page 18: https://www.utusan.com.my/category/gaya/sastera/page/18/
Scraping page 19: https://www.utusan.com.my/category/gaya/sastera/page/19/
Scraping page 20: https://www.utusan.com.my/category/gaya/sastera/page/20/
Failed to scrape page 20 of sastera: 404 Client Error: Not Found for url: https://www.utusan.com.my/category/gaya/sastera/page/20/
--- Scraping subcategory: SERAM ---
Scraping page 1: https://www.utusan.com.my/category/pancaindera/seram/
Scraping page 2: https://www.utusan.com.my/category/pancaindera/seram/page/2/
Scraping page 3: https://www.utusan.com.my/category/pancaindera/seram/page/3/
Scraping page 4: https://www.utusan.com.my/category/pancaindera/seram/page/4/
Scraping page 5: https://www.utusan.com.my/category/pancaindera/seram/page/5/
Failed to scrape page 5 of seram: 404 Client Error: Not Found for url: https://www.utusan.com.my/category/pancaindera/seram/page/5/
--- Scraping subcategory: POLITIK ---
Scraping page 1: https://www.utusan.com.my/category/nasional/politik/
Scraping page 2: https://www.utusan.com.my/category/nasional/politik/page/2/
Scraping page 3: https://www.utusan.com.my/category/nasional/politik/page/3/
Scraping page 4: https://www.utusan.com.my/category/nasional/politik/page/4/
Scraping page 5: https://www.utusan.com.my/category/nasional/politik/page/5/
Scraping page 6: https://www.utusan.com.my/category/nasional/politik/page/6/
Scraping page 7: https://www.utusan.com.my/category/nasional/politik/page/7/
```

UtusanMalaysia_selectolax.csv

Title	URL	Category	Date
Aura Walid â€˜one takeâ€™	https://www.utusan.com.m	cover	27 April 2025, 9:00 am
Terumbang ambing mencipta nama -Sheila Abdull	https://www.utusan.com.m	cover	20 April 2025, 6:50 am
Terima kasih Maya pinjamkan â€˜Babahâ€™ untuk saya	https://www.utusan.com.m	cover	13 April 2025, 9:00 am
â€˜Daripada jawab soal jodoh, saya rela duduk dapurâ€™ â€“	https://www.utusan.com.m	cover	6 April 2025, 6:50 am
Memori raya dengan Dee? Hanya Allah yang tahu- Farid Kamil	https://www.utusan.com.m	cover	30 Mac 2025, 8:00 am
Dulu ada yang kata muzik Mohram tak boleh jual	https://www.utusan.com.m	cover	23 Mac 2025, 9:10 am
â€˜Kami berbeza agama, bukan semua orang sukaâ€™	https://www.utusan.com.m	cover	23 Mac 2025, 6:50 am
Kalau semua betul, kita tidak belajar apa-apa â€“ Mira Filzah	https://www.utusan.com.m	cover	16 Mac 2025, 8:00 am
Datanglah berjumpa orang yang dia zalimi ini	https://www.utusan.com.m	cover	9 Mac 2025, 7:50 am
Tiada lagi gulai ikan talang ibu	https://www.utusan.com.m	cover	2 Mac 2025, 9:00 am
Saya menangis â€˜mencariâ€™ Tracie Sinidol	https://www.utusan.com.m	cover	23 Februari 2025, 8:00 am
Bagai ada beban yang harus dipikul	https://www.utusan.com.m	cover	16 Februari 2025, 9:00 am
Liza Aziz lunas impian setelah lebih empat dekad	https://www.utusan.com.m	cover	9 Februari 2025, 8:00 am
Umai mahu bersinar dengan identiti sendiri	https://www.utusan.com.m	cover	2 Februari 2025, 6:17 am
Penat lelah empat dekad terbayar	https://www.utusan.com.m	cover	26 Januari 2025, 6:39 am
Hun Haqeem pakai mask naik LRT lihat manusia	https://www.utusan.com.m	cover	19 Januari 2025, 6:21 am
Popularitl tidak pernah jatuh menjunam	https://www.utusan.com.m	cover	12 Januari 2025, 6:33 am
Puaskan hati semua orang? Saya tidak akan mampu â€“ Noki	https://www.utusan.com.m	cover	5 Januari 2025, 6:40 am
â€˜Terima kasih Siti, si gadis bergaun merahâ€™	https://www.utusan.com.m	cover	29 Disember 2024, 6:29 am
Dulu saya hanya mampu makan telur sahaja â€“ Nadeera Zai	https://www.utusan.com.m	cover	22 Disember 2024, 6:09 am
â€˜Saya korbankan usia remaja, demi muzikâ€™	https://www.utusan.com.m	cover	15 Disember 2024, 6:11 am
Sebar manfaat sebelum menutup mata	https://www.utusan.com.m	cover	8 Disember 2024, 6:23 am
Yang muda kena perbaiki salah faham ini â€“ Harissa Adlynn	https://www.utusan.com.m	cover	1 Disember 2024, 6:30 am
Pernah bernyanyi di kafe sebelum popular	https://www.utusan.com.m	cover	24 November 2024, 6:50 am
â€˜Aktor paling mahal? Bukan sayaâ€¦â€™	https://www.utusan.com.m	cover	10 November 2024, 7:45 am

Data Processing and Optimization Output

Section	Screenshot
Pure Python (Pandas)	Performance Report [Pure Python] Final Row Count: 126,294 Time Taken: 2.3289 seconds Throughput: 54228.29 rows/sec CPU Usage: 98.50% Memory Used: 1524.07 MB
Polars	Performance Report [Polars (Optimized)] Final Row Count: 126,294 Time Taken: 2.3170 seconds Throughput: 54507.11 rows/sec CPU Usage: 86.40% Memory Used: 1537.03 MB
Modin	Performance Report [Modin] Final Row Count: 126,294 Time Taken: 1.0832 seconds Throughput: 116595.69 rows/sec CPU Usage: 16.80% Memory Used: 1531.86 MB
Dask	Performance Report [Dask] Final Row Count: 126,294 Time Taken: 1.2191 seconds Throughput: 103598.93 rows/sec CPU Usage: 75.40% Memory Used: 1533.86 MB

Swifter	Performance Report [Swifter] Final Row Count: 126,294 Time Taken: 1.4832 seconds Throughput: 85149.55 rows/sec CPU Usage: 36.90% Memory Used: 1532.44 MB
Save Cleaned Data and Load to MongoDB	Saved: cleaned_python.csv Uploaded to MongoDB collection: cleaned_python_data Saved: cleaned_polars.csv Uploaded to MongoDB collection: cleaned_polars_data Saved: cleaned_modin.csv Uploaded to MongoDB collection: cleaned_modin_data Saved: cleaned_dask.csv Uploaded to MongoDB collection: cleaned_dask_data Saved: cleaned_swifter.csv Uploaded to MongoDB collection: cleaned_swifter_data
Comparison in Bar Charts	
Comparison Table	Comparison Table (Performance Metrics by Library): Library Final Row Count Execution Time (s) CPU Usage (%) Memory Usage (MB) Throughput (rows/sec) Pure Python 126294 2.328932 98.5 1524.066406 54228.288957 Polars 126294 2.317019 86.4 1537.031250 54507.106233 Modin 126294 1.083179 16.8 1531.855469 116595.687643 Dask 126294 1.219067 75.4 1533.863281 103598.932118 Swifter 126294 1.483202 36.9 1532.441406 85149.548984

Links to Full Code Repo

Github

https://github.com/Jingyong14/HPDP02/tree/a6d028845e5040da68b99a245802950ec122314c/2425/project/p1/Group%203