SECP3133-02

HIGH PERFORMANCE DATA PROCESSING

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

Prepared By:

VINESH A/L VIJAYA KUMAR A22EC0290

JOSEPH LAU YEO KAI A22EC0055

TIEW CHUAN SHEN A22EC0113

NUR FARAH ADIBAH BINTI IDRIS A22EC0245

Lecturer Name:

DR. ARYATI BINTI BAKRI

Date of Submission:

28/5/2025

Table of Contents

1. Introduction

1.1 Background of the project

The demand for data analytics in real time and low latency is growing nowadays to meet the needs of the market, which high performance data processing is focused on across various industries. High performance data processing emphasizes the use of enhanced computational methods such as high performance computing(HPC) to perform data processing processes such as data collection, cleaning and analysis in a short time. The project is focused on answering:

1. Does high performance computing(HPC) infrastructure and methods enhance the performance of the data processing phase?
2. How is the performance of different Python libraries and frameworks (Pandas, Dask) in implementing HPC for data processing tasks?

Through this research, the project is aimed to provide comparative analysis on the impact of different libraries to web scraping, data cleaning and analysis.

1.2 Objectives

1. To develop web crawlers with different libraries that are able to extract at least 100,000 records from the News Straits Times (NST) website.
2. To store extracted data in MongoDB for further processing.
3. To clean and preprocess the raw dataset.
4. To evaluate performance before and after optimization using several performance metrics.

1.3 Target website and data to be extracted

In this project, the targeted website is New Strait Times(NST), with the link www.nst.com.my. New Strait Times or NST is one of Malaysia's most known news publishers in English. Various domains are offered by New Strait Times such as national news, business, politics, sports and lifestyle. The platform is providing a huge dataset of articles, enabling the website to be a good source for data analytics. NST is selected for its huge data volume and consistent news structure and format which allows for the smooth extraction process for the project. The articles provide metadata and content sections which are suitable for web crawling.

The main focus will be about extracting the informations from individual news articles from different section, which key data attributes targeted are shown in table 1 below:

No	Data Field	Data Type	Description
1	Section	String	News topic(crime, politics, nation, health).
2	Publication date	Date	The date(including time) the article is published with format mm:dd:yyyy @ hh:mm
3	Headline	String	Title of the article.
4	Summary	String	Brief summary of the news.

Table 1: Data attributes planned to be scrapped from website

Through these attributes, a valuable dataset will be collected for analyzing and researching insights. The crawling process will be designed with respects to ethical scraping practices and rules, to avoid adding great workloads to the web server through appropriate delays between requests.

2. System Design & Architecture

This section explains the systematic method and approach used to design and implement the extraction of raw data from the website, data cleaning to the process of export cleaned dataset to the database. The overall architecture, tools and frameworks and roles of each team member are explained in detail. The design and architecture focuses on ensuring the whole process is well- structured and efficient.

2.1 Architecture

The architecture of the whole process is started with the web scraping, which gathers news article information from the News Strait Time(NST) website. Web crawlers are developed using Selenium and BeautifulSoup libraries, which Selenium is specified for navigating dynamic web content and to handle the JavaScript rendered elements. BeautifulSoup is used for parsing HTML content and structure. The NST website is scrapped based on the link below https://www.nst.com.my/news/topics?pages. Crawlers are made specifically by teammates for different news topics, including Politics, Crime and Court, Government and Nation, in order to collect a wide variety of data from the NST website. For each article, the data attributes including the Section, Publication Date, Headline and Summary are extracted from the website. To follow the ethical scraping practices, rate limitation is implemented and robots.txt are followed while scraping each page to prevent overloading the website server.

After the raw data is extracted from the website, the data is formatted and saved as the Comma-Separated Values(CSV) files. The complete raw dataset is then loaded into MongoDB database for data cleaning and transformation process later. MongoDB is selected for schema flexibility to deal with the scraped data which may have different structure. Storing the raw data in MongoDB allows teammates to manage data in a centralized source and ensure the data availability at all times.

The next step is data preprocessing, starting with the process of basic data cleaning using Pandas library. Raw data is retrieved from MongoDB and several data preprocessing processes are considered for cleaning the data. The processes include handling missing value in dataset, removing duplicates for redundant data and format standardizing such as converting the Section column to Title case and changing the date time to standard date time format. The cleaned data is then stored as a new collection in MongoDB, named as cleaned_data_Pandas. This process act as a benchmark for comparing with the performance of the other libraries.

The next stage focuses on the high performance data cleaning through the application of the libraries including Polars, Vectorized Pandas, PySpark and Dask. The raw data from MongoDB is processed through these four libraries for comparing the performance between them. The cleaned data is saved as a new collection in MongoDB using the naming format, cleaned_data_library. In the whole process, the aspects such as execution time, CPU usage, peak memory usage and throughput are measured for each library for comparison evaluation. This comprehensive architecture not only shows the whole process from web scraping to data

cleaning using different libraries, but also provides a framework for understanding the impact of different high performance data processing tools.

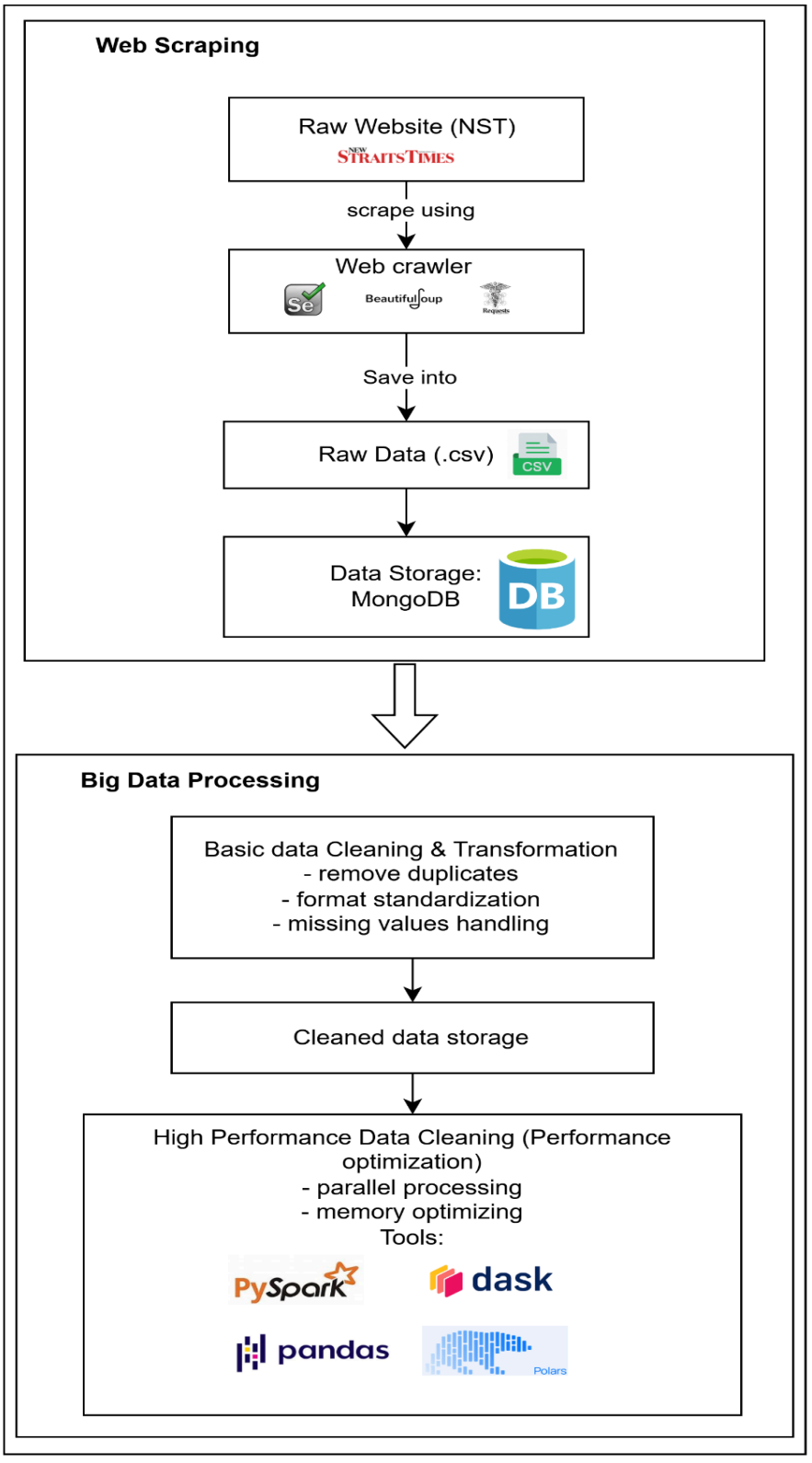


Figure 2.1: System Architecture for web scraping and performance evaluation

2.2 Tools and Framework used

The project selected several key tools to manage the workflow efficiently from web scraping to performance analysis as shown in Table 2.2. Python is the main language, with Selenium and BeautifulSoup for web scraping. MongoDB is used for data storage for raw data and cleaned data. For data preprocessing and optimization, Pandas is acting as baseline, while Dask, Polars, PySpark and Vectorized Pandas are used for high performance data processing. The whole development environment is performed in the Google Colab and Visual Studio Code, as well as the Draw.io for diagram design and Google Docs for reporting.

Category	Software/Application/Library/Website
Reporting	Google Docs
System Architecture and Design	Draw.io
Web Scraping Library	BeautifulSoup, Selenium
Data Storage	CSV load to MongoDB
IDE	Google Colab, Vs Code
Coding Language	Python
Data Cleaning+Performance Evaluation	Python
Libraries used in process optimization	PySpark, Polars, Vectorized Pandas, Dask

Table 2.2: Tools and framework used

2.3 Roles of team members

To ensure the efficient project execution, each teammate is assigned with different responsibilities as shown in Table 2.3 below.

Member	Role	Responsibility
Joseph Lau Yeo Kai	Web Scraping Process Designer	Design and develop whole web scraping code, at the same time ensures robots.txt and ethical data scraping is followed.
Nur Farah Adibah Binti Idris	Data Processing Specialist	Design and develop data cleaning process based on the data scraped from NST website. Ensure the cleaned data is consistent for each library used.
Vinesh A/L Vijaya Kumar	Process Efficiency Tester	Ensure resource usage and optimizes processing speed for web scraping and data cleaning.
Tiew Chuan Shen	Data Visualization & Reporting	Design evaluation metrics and code, for producing visual reports in graphs. Combines all documentation and report for submission.

Table 2.3: Team members' roles

3. Data Collection

In this phase, there are about 127729 rows of data being scraped from the New Straits Times(NST) website, with each row of data containing attributes- section, date, headline and summary. We develop web crawlers with several libraries such as Scrapy, BeautifulSoup and Selenium, which are used for browsing through pages and handling pagination efficiently. After trial and error, we find out that the best solution for web scraping is the combination of BeautifulSoup and Selenium, which can navigate and browse through pages,

To follow the ethical standards and avoid overloading the server of the NST website, we follow the robots.txt of the NST website, and rate limitation is included in our web scraping process design through adding sleep time(delays) upon each page scraped. Besides, we ensured that only one person will be allowed to perform web scraping at the same time.

3.1 Crawling Method

The web crawling method implemented for the New Straits Times website is important to extract news efficiently and in a structured manner from specific sections. There were two methods used to carry this out including Pagination Handling and Rate Limiting.

The pagination handling was necessary to scrap news articles available in various sections of the NST website. The crawlers were designed to navigate step by step from page to page of listings within specified news categories like 'Crime-courts' and 'Nation'. This was usually done by finding and utilizing the URL pattern NST follows for pagination, usually involving modifying a URL parameter (e.g., ?page=X, where 'X' is the page number) to retrieve further pages of articles which can be refer through figure 3.1.1 below

```
def main():
    news_cat = ['crime-courts']
    all_articles = []

        driver = setup_driver()
    try:
        for cat in news_cat:
            news_path = f'/news/{cat}'
            full_path = urljoin(BASE_URL, news_path)

            for page in range(700,900):
                page_url = f"{full_path}?page={page}"
                articles_data = scrape_nst_articles(page_url, driver)
                all_articles.extend(articles_data)
```

Figure 3.1.1 Pagination handling in web crawler

The system also had error handling mechanisms in place to deal efficiently with cases when a page would not load properly or when the end of pagination in a section was reached, preventing the unanticipated crashes of the crawler. This methodical process helped us achieve good coverage for articles in the given categories. In trying to uphold web scraping ethics and also to avoid overloading the NST web servers, the implementation of Rate

Limiting was very important to the crawling process. A timed delay of five seconds between each request for a new page of articles was introduced which can be referred to in the figure 3.1.2 below.

```
time.sleep(5)  # Rate limiting
```

Figure 3.1.2: Rate Limiting method used in web scraping

This slow pacing is intended to prevent the target site from being swamped with too many requests within a short time and out of respect for the limited resources of the website and minimising the likelihood of being blocked or interfering with the normal operations for other users. This careful consideration of the request rate was a part of the project's ethical approach to data collection.

3.2 Number of Records Collected:

127,729 news articles are extracted from the New Straits Times website. Every record in this dataset is formatted to contain four core data fields: the 'Section' the article is associated with (such as Crime-courts or Nation), the 'Date' of the article, the article 'Headline', and a brief 'Summary' of content.

This vast amount of data was collected by running multiple sessions of scraping at various time intervals, with the aim of collecting a broad and representative sample of news articles for processing and analysis. Successful collection of this huge dataset provides the basis upon which the efficacy of different data processing optimization methods can be measured. Followed by Figure 3.2: Outcome of the data collection process stored in CSV format as indicated in the initial document.

Section	Date	Headline	Summary
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	KL police confirm receiving Istana Negara's report over fake letter to King	KUALA LUMPUR: City police confirmed receiving a report from Istana Negara's representative over a letter allegedly written by Foreign Minister Datuk Seri Hishammuddin Hussein to ap
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Datuk and brother charged with making false claims for cooking oil subsidies	KUALA LUMPUR: A man with a 'Datuk' title and his brother were slapped with 52 counts of filing forged documents for subsidy payments totalling RM1.5 million, since two years ago.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Deputy, asst director charged with using false receipts for claims	KOTA KINABALU: A deputy and assistant director from the Malaysian Communication and Multimedia Commission (MCMC) were separately charged in the Sessions Court here with se
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	13 charged with MCO violation by participating in funeral procession [NSTTV]	
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Sergeant remanded over RM31,000 graft [NSTTV]	GEORGE TOWN: A police sergeant has been remanded for six days in connection with a RM31,000 graft case.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Man arrested for possessing protected birds	JOHOR BARU: The Wildlife and National Parks Department (Perhilitan) detained a man for possessing protected wild birds at a house in Ulu Tiram, here.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	11 held in drug party at Bukit Mertajam homestay	BUKIT MERTAJAM: 11 people, including five women, were detained during a drug-fuelled party at a homestay in Kompleks BM City here past midnight today.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	GOF foils attempts to smuggle subsidised items into Thailand	TUMPAT: The General Operations Force (GOF) foiled two attempts to smuggle Malaysian subsidised cooking oil, flour and sugar into Thailand with the seizure of the items worth abou
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	CITF lodges 2 police reports over sales of Covid vaccines	KUALA LUMPUR: The Covid-19 Immunisation Task Force (CITF) has lodged two police reports, one on June 14 and another on June 22 following public complaints on the selling activiti
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Three nabbed for selling Covid-19 vaccines	KUALA LUMPUR: Police arrested three people in Putrajaya and Brickfields here yesterday for allegedly being involved in the sale of Covid-19 vaccines.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Cops hunting man who sexually assaulted two children	PETALING JAYA: Police have launched a hunt for a man who is believed to have sexually assaulted two children at a block of flats in Section 19 here last Monday.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Jobless man jailed for trying to steal bank aircon compressor	Ersie Anjumin
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Two Pakistani men fined RM1,500 each for flouting MCO SOP	SELAYANG: Two Pakistani men who were caught not wearing face masks in front of a factory in Gombak were both fined RM1,500 each by the magistrate's court here today.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Police bust illicit cigarette trafficking syndicate in Tasek Gelugor	KEPALA BATAS: Bukit Aman Internal Security and Public Order Department (KDNKA) foiled an illicit cigarette trafficking syndicate with the seizure of 18,490,000 sticks of cigarettes wort
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Bung Moktar, Zizie Izette's corruption trial to resume July 26	KUALA LUMPUR: The corruption trial of Datuk Seri Bung Moktar Radin and his celebrity wife Datin Seri Zizie Izette A Samad will resume on July 26 due to the ongoing Movement Control
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	28 cows worth about RM282,000 seized at Kelantan border	PASIR MAS: A 21-year-old lorry driver was arrested in possession of 28 heads of smuggled cattle worth about RM282,000 here this morning.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Game over for 10 caught gambling in bushes behind Lipis temple	LIPIS: A makeshift gambling den among bushes at a rubber plantation behind a temple at Kampung Baru Penjom near Kuala Lipis here did not stay hidden for long.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Man caught red-handed attempting to break CDM machine	BALIK PULAU: A man was caught red-handed trying to break a cash deposit machine (CDM) at an AmBank branch along Lintang Bukit Penara here last night.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	754 people arrested for violating SOP	KUALA LUMPUR: A total of 754 people were arrested for violating the Movement Control Order (MCO) standard operating procedures (SOP) yesterday.
CRIME & COURTS	Jun 25, 2021 @ 3:59pm	Man nabbed after driving 10km in stolen lorry	GEORGE TOWN: A man was detained shortly after he drove off in a lorry which he stole from the parking lot of a hypermarket in Bayan Baru here, this afternoon.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Man nabbed after driving 10km in stolen lorry	GEORGE TOWN: A man was detained shortly after he drove off in a lorry which he stole from the parking lot of a hypermarket in Bayan Baru here, this afternoon.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	152 bookies nabbed over betting in Euro Cup matches	
CRIME & COURTS	Jun 23, 2021 @ 11:08am	One nabbed, another escapes in 1km police chase in Kepala Batas	KEPALA BATAS: A man managed to escape while his friend was nabbed by police during a 1km-chase in Penaga here yesterday.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Doctor fined RM5,000 for publishing fake vaccine news	KANGAR: A doctor was fined RM5,000 by the Sessions Court here today, in default five months' jail, for publishing a video containing fake news about Sinovac vaccine in April.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Man arrested for threatening woman who supported student	KUALA LUMPUR: Police have arrested a man believed to have threatened a woman who showed support to Ain Husniza Saiful Nizam on Twitter last month.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Duo fined for harbouring illegal immigrants	KOTA KINABALU: Two 46-year-old locals were fined for allowing illegal immigrants to stay at their premises.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Man charged with trying to stab policeman	KOTA BELUD: An unemployed man who tried to stab a police corporal with a knife three years ago, was charged at the magistrate's court here today.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Five-day remand for father who allegedly fed baby alcoholic drink	BALIK PULAU: Police have obtained a five-day remand order against a young father who allegedly fed his infant son liquor straight from the can.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	5, including 3 women, detained in Pasir Mas over syabu, heroin posession	PASIR MAS: Three women and two men were arrested for possession of heroin and syabu during a raid at Kampung Banggol Petai, here, yesterday.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Cleaner in hit-and-run case faces murder, obstructing police, drug charges [NSTT	GEORGE TOWN: A cleaner, who was detained last Tuesday after police were forced to shoot at his rented Perodua Myvi following a fatal hit-and-run accident, has been charged with f
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Man arrested with about RM28,000 worth of ketum leaves	PASIR MAS: Police detained a man and seized 80 plastic bags of ketum leaves worth about RM28,000 in an operation here today.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Police nab man who fed baby alcoholic drink	BALIK PULAU: Police have detained a 19-year-old man in connection with a 30-second video of him allegedly feeding a baby liquor straight from the can.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Pregnant woman to know fate in Nov after pleading guilty to 6 counts of bribery	KOTA KINABALU: A pregnant woman pleaded guilty to six counts of abusing her position to obtain bribes at the Special Corruption Court, here.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	July 16 for decision on IGP's bid to strike out Indira Gandhi's suit	KUALA LUMPUR: The High Court has fixed July 16 for the decision on the Inspector-General of Police's (IGP) bid to strike out a lawsuit by M. Indira Gandhi.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	NGO wants police to act against man who fed alcoholic drink to baby	GEORGE TOWN: Malaysian Anti-Cheap Liquor Movement (MACLM) president David Marshel lodged a police report at the Prai police station this afternoon in connection with a 30-secc
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Sakai Gang leader, 23 members arrested in four states [NSTTV]	KUALA LUMPUR: Police arrested the infamous Sakai Gang leader together with 23 other gang members after raids in Melaka, Negri Sembilan, Terengganu and Perak on Sunday.
CRIME & COURTS	Jun 23, 2021 @ 11:08am	Sabah MACC nabs man who bribed authorities in relation to gambling activities	TAWAU: The Sabah Malaysian Anti-Corruption Commission (MACC) has nabbed a man who had been bribing enforcement personnel in the district for years to conduct illegal gamblii
CRIME & COURTS	Jun 23, 2021 @ 11:08am	12th man held over funeral procession; police looking for two others	BUTTERWORTH: Police have detained yet another man in connection with a viral video footage of a group of men who were seen carrying a casket along Jalan Siram while others danc

Figure 3.2 Result of data collection process stored in csv

3.3 Ethical Considerations

During the large-scale data collection, special attention was given to maintaining ethical web crawling practices to practice a responsible and respectful engagement with the New Straits Times website. The deployment of the web crawlers encompassed several key considerations to address these ethical demands:

1. Server Load Management: Delays (rate limiting) were purposely placed between consecutive page requests. This crucial step was taken to prevent overloading the NST web servers, thus ensuring that the scraping activities did not impact the performance or availability of the website for other users.
2. Fewer Resource Consumption: Headless browser setup was employed wherever possible throughout the whole process of web scraping. This strategy reduces the computation resources drawn by the crawler from the client side and can additionally lighten the load on the target server in contrast to the full graphical browser session.
3. Error Handling: The web spiders were built with error-handling mechanisms for dealing with problems like network loss or unreachable pages. This approach minimized cases of excessive or repetitive attempts to retry problematic URLs, thereby lessening unnecessary server load.
4. Purely Publicly Available Information: The data collection parameters were strictly limited to information that is publicly accessible on the NST website. No attempts were made to collect private, restricted, or sensitive information.
5. Robots.txt adherence: The directives given in the robots.txt file of the New Straits Times website were examined thoroughly and adhered to strictly. This permitted the crawlers to visit just those sections of the website that the publisher allows for automated retrieval.

4. Data Processing

The data processing implementation is focused on data cleaning, transformation and storage using different libraries- Pandas, PySpark, Dask and Polars, for performance comparison and evaluation. Raw data is loaded from MongoDB for the data cleaning process.

4.1 The cleaning and transformation methods

Data cleaning and transformation methods are applied based on data inconsistencies as below:

1. Handling Null Values: Rows that had missing or null values in any of the important columns (Section, Date, Headline, Summary) were located. In order to preserve data integrity, these rows were completely eliminated from the dataset through a dropna() operation.
2. Eliminating Duplicate Values: The data was tested for duplicate records, with whole rows (all four attributes the same) being duplicated. Duplicates of this nature can affect the analysis result. A drop_duplicates(keep='first') function was utilized, which only kept the first instance of any duplicate record and removed all following duplicates.
3. Standardizing 'Section' Column: 'Section' column, representing the news type, previously had inconsistent capitalization (e.g., "crime", "Crime", "CRIME"). For consistency, all values in this column were standardized to title case format (e.g., "Crime") with string manipulation functions (.str.title()).
4. Cleaning and Formatting 'Date' Column: The 'Publication Date' column, upon scraping, usually had some extra time information (e.g., "@ hh:mm") and also differed slightly in string format. Cleaning was done by eliminating the date portion by splitting on the "@" character and only keeping the date component. Any stray whitespace was also eliminated. Following this string cleaning, the column was converted to a standardized datetime object (i.e., YYYY-MM-DD) through pd.to_datetime(), with any parse errors forced into NaT (Not a Time) values, which could then be addressed if necessary (though the initial dropna would likely eliminate these).

Table 4.1 below is the full code based on the above data inconsistencies and their solutions.

Code	Explanation

# Drop nulls and duplicates 	
df = df.dropna()
df = df.drop_duplicates(keep='first')
	Delete the rows which are duplicated or have null value.

# Standardize 'Section' column if 'Section' in df.columns: df["Section"] = df["Section"].str.title()	Change all data in section column into Title format.

# Clean 'Date' column 	
def clean_date(date):
if isinstance(date, str):
if "@" in date:
date =
date.split("@")[0].strip()
return re.sub(r'\s+', ' ', date)
return date
 	
if 'Date' in df.columns:
df["Date"] =
df["Date"].apply(clean_date)
df["Date"] =
pd.to_datetime(df["Date"],
errors='coerce')
	Change the format of date into formal format.

Table 4.1: Code overview Data Processing

4.2 Data structure (Database)

The raw data and cleaned dataset are all stored in MongoDB. Data is called and stored as shown in Figure 4.2 below.

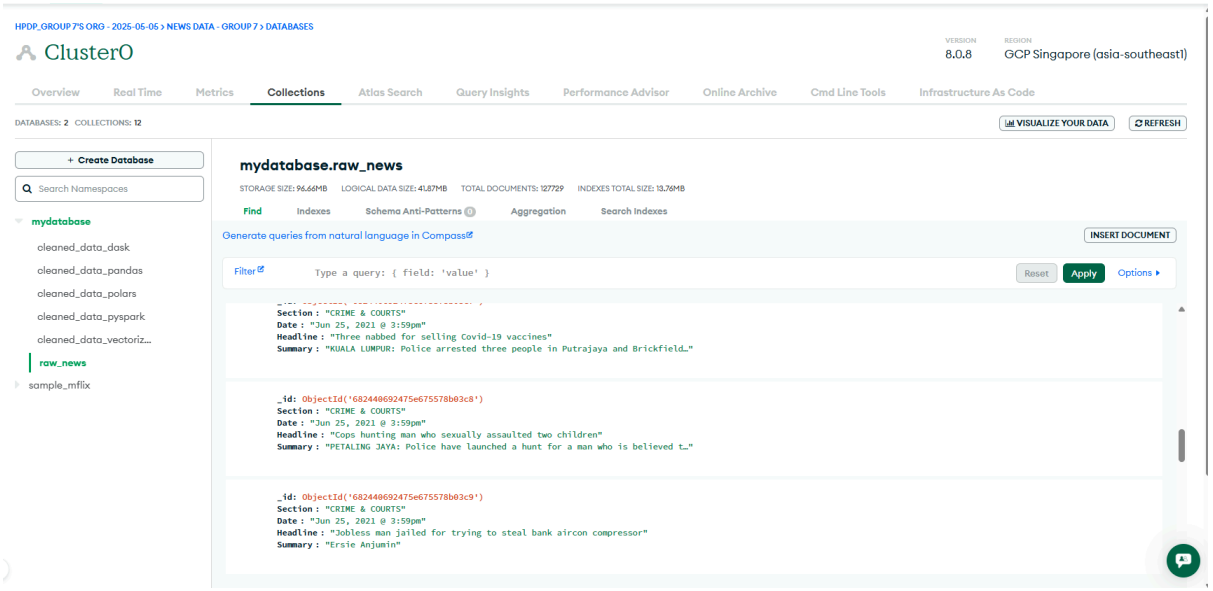


Figure 4.2 MongoDB as data storage for raw and cleaned data

5. Optimization Techniques

This section introduces the optimisation method used for faster data processing and shows an overview code for each technique.

5.1 Methods used

In this project, the initial implementation for data processing uses the basic pandas library. To optimise the performance of the data processing, we use Polars, Dask, PySpark and Vectorised Pandas. The table below explains these optimization techniques.

Table 5.1: Explanation for each optimization technique

Library / Technique	**Explanation on optimisation**
Polars	Built on Rust instead of Python and uses columnar data storage for faster data retrieval and more efficient processing,
Dask	Extends Pandas functionality by processing in parallel and lazily. In the optimisation code, map_partitions are used to divide datasets into partitions for parallel operations.
Pyspark	Uses Apache Spark for distributed data processing.
Vectorised Pandas	Eliminate .apply() in basic pandas code and change it into vectorised functions. Vectorised pandas use Pandas functions that operate on entire columns instead of using .apply() and loops.

5.2 Code overview of techniques applied

The code overview demonstrates different optimization ways of cleaning the ‘Date’ Column, the uncleaned ‘Date’ Column format is String “Jun 25, 2021 @ 3:59pm”, processed into datatype Date 2021-06-25.

Table 5.2: Code overview for each optimization technique

Technique	**Code overview for ‘Date’ column processing**	**Explainantion**

Polars	if 'Date' in df.columns:
	df = df.with_columns(
	df['Date']
	.str.split('@')
	.list.first()
	.str.strip_chars()
	.alias('Date')
	)
	
	# Convert to datetime
	df=df.with_columns(pl.col('Date').str.strptime(pl.Datetime, format='%b %d, %Y', strict=False).alias('Date'))
		Eliminate the unnecessary part by splitting the string at ‘@’ and taking the first part. Then strips unwanted characters and converts the string to a datetime using Polar's function.

Dask	df_cleaned = df_cleaned.map_partitions(lambda df: df.assign(Date=df['Date'].map(lambda x: x.split('@')[0].strip() if isinstance(x, str) else x))) df_cleaned = df_cleaned.map_partitions(lambda df: df.assign(Date=pd.to_datetime(df['Date'], errors='coerce')))	Using map_partitions to apply operations on each partition (can refer to the appendices, we used four partitions). The operations include splitting at ‘@’, stripping whitespace and parsing to datetime.
Pyspark	df_cleaned = df_cleaned.withColumn("Date", regexp_replace(col("Date"), "@.*$", "")) df_cleaned = df_cleaned.withColumn("Date", regexp_replace(col("Date"), "\s+", " ")) df_cleaned = df_cleaned.withColumn("Date", trim(col("Date"))) df_cleaned = df_cleaned.withColumn("Date", to_date(col("Date"), "MMM d, yyyy"))	Using Spark functions to remove the part after ‘@’, normalise spaces, trim and convert to date format.
Vectorised Pandas	if 'Date' in df.columns: df['Date'] = df['Date'].str.split('@').str[0].str.strip() df['Date'] = df['Date'].str.replace(r'\s+', ' ', regex=True) df['Date'] = pd.to_datetime(df['Date'], errors='coerce')	Using vectorised string and datetime operations in pandas for processing the ‘Date’ column without iteration and .apply() function.

6. Performance Evaluation

6.1 Before vs after optimization

Initially, we would process the data with plain Pandas, which was decent for small data but would take forever to execute with large-scale datasets, particularly when filtering, aggregating, and joining. To address such shortcomings and boost performance, we introduced and experimented with some high-performant alternatives: Vectorized Pandas, Dask, Polars, and PySpark.

•**Vectorized Pandas** enhanced standard Pandas operations by eliminating slow.apply() and loop constructs and instead using native, column-based operations. This yielded dramatic speed gains with minimal alteration of code.

•**Dask** extended Pandas with parallelism and lazy evaluation, allowing for efficient handling of datasets that do not fit into memory, by partitioning the data and processing in parallel.

•**Polars**, which was implemented with Rust having a multi-threaded, columnar backend, offered the optimum performance for in-memory data processing operations. It greatly outperformed others both in speed and memory usage.

•**PySpark**, while heavier due to its distributed architecture, provided consistent performance for extremely large data sets. While experiencing some initialization overhead, it performed reasonably well in a local distributed environment.

According to this preliminary text's conclusion, execution times for all tasks greatly decreased after implementing these changes. It points out that Polars outperformed Dask and Vectorized Pandas in terms of speed and memory utilization, particularly on smaller to medium-sized datasets. Although PySpark performed well on distributed loads, it was slower in several situations.

6.2 Comparison of Code Execution Time, Peak Memory Usage, CPU usage and Throughput

Table 6.1: Comparison between Data Processing and Cleaning Techniques

Operation	Aspects	Comparisons				
		Dask	Polars	Pyspark	Pandas	Vectorized Pandas
Dataset Loading and Display	Code Execution Time (s)	0.5574	0.13728	0.1715	1.45037	0.81039
	Peak Memory Usage (MB)	9.566	0.3828	0.0	0.0	0.0
	Throughput (rows/s)	217847.737	884573.6415	707961.62	83728.7675	149849.72

Essentially, Table 6.1 provides numerical support for the assertions stated in the previous paragraph. Polars continuously performs at the highest level in terms of throughput and execution time while using little memory. Additionally, PySpark has excellent throughput and execution time performance, especially with a small memory footprint. Pandas are typically the slowest and least effective when vectorization is not used. For this particular process, Dask uses more memory than Polars, PySpark, and Vectorized Pandas, despite being an improvement over ordinary Pandas.

Conclusion:

Polars > Vectorized Pandas > Dask > Pandas > PySpark

•**Polars:** Offers the overall best performance in every category. With its Rust foundation, multi-threaded operation, and columnar in-memory storage, it was able to handle massive volumes of data with extremely low memory utilization and extremely high rates of computation.

•**Vectorized Pandas:** Displayed extensive improvement over traditional Pandas by embracing efficient, column-based operations. It provided sturdy throughput with zero memory usage, which made it extremely effective for medium-sized datasets without parallel or distributed systems.

•**Dask:** Performed better than standard Pandas by allowing parallelism and chunked data processing. Though slower than Vectorized Pandas for this application, Dask excelled on the scaling aspect and was memory-friendly.

•**Pandas:** Although easy to manipulate, Pandas showed large memory usage and sluggish run times with increasing data size. It was the least scalable but set a good baseline.

•PySpark: Registered worst performance during this test in local-mode mode due to too much initialization overhead and resource consumption. But it remains a strong candidate for scaled distributed processing, particularly in multi-node or cluster modes.

6.3 Charts and graphs

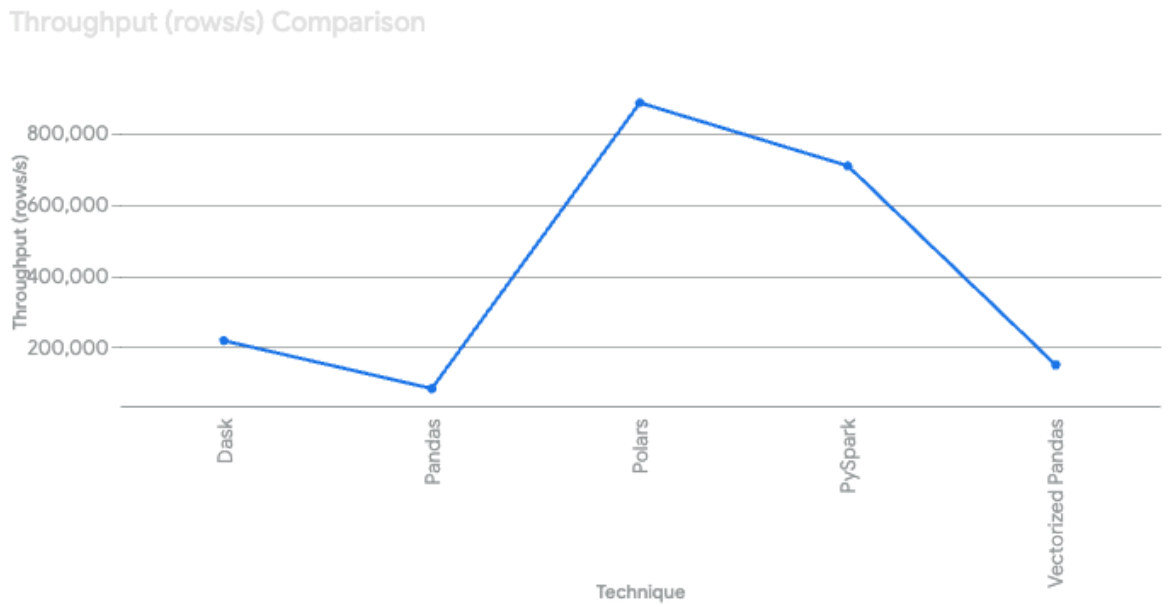


Figure 6.3.1: shows Throughput(rows/s) of each optimization

This line graph, called "Throughput (rows/s) Comparison," shows how well different methods perform by counting the number of rows that are processed in a second. The X-axis shows five distinct techniques: Dask, Pandas, Polars, PySpark, and a combination Vectorize Pandas. The Y-axis shows the throughput in rows per second, which ranges from 200,000 to 800,000. While "Pandas," "Polars," and "Vectorize Pandas" all exhibit varying degrees of improved throughput in between, the graph shows that the "PySpark" technique achieves the highest throughput, approaching 800,000 rows per second, greatly surpassing the "Dask" technique, which has the lowest throughput of about 200,000 rows per second.

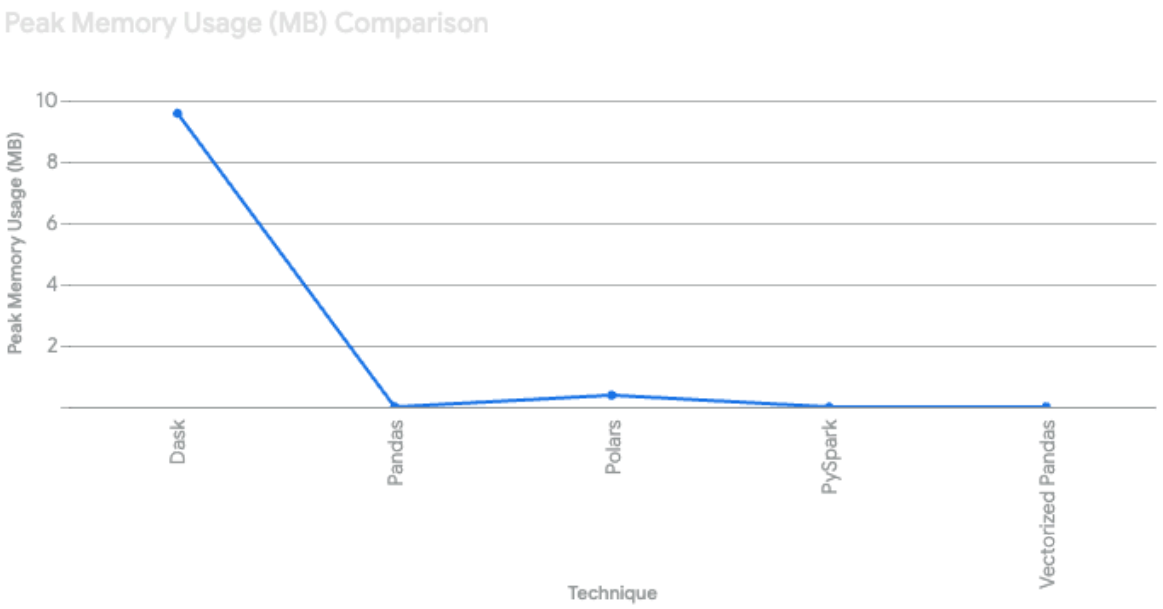


Figure 6.3.2: shows Peak Memory Usage(MB) of each optimization

This line graph, called "Peak Memory Usage (MB) Comparison," shows the maximum memory used by different techniques. The Y-axis shows peak memory usage in megabytes, which ranges from 0 to 10 MB. The X-axis lists the techniques: Dask, Pandas, Polars, PySpark, and a combination of Vectorize Pandas. The graph clearly shows that the "Dask" technique utilizes the highest peak memory, nearing 10 MB, while all subsequent techniques, including "Pandas," "Polars," "PySpark," and "Vectorize Pandas," demonstrate significantly lower and more efficient memory usage, generally remaining below 2 MB.

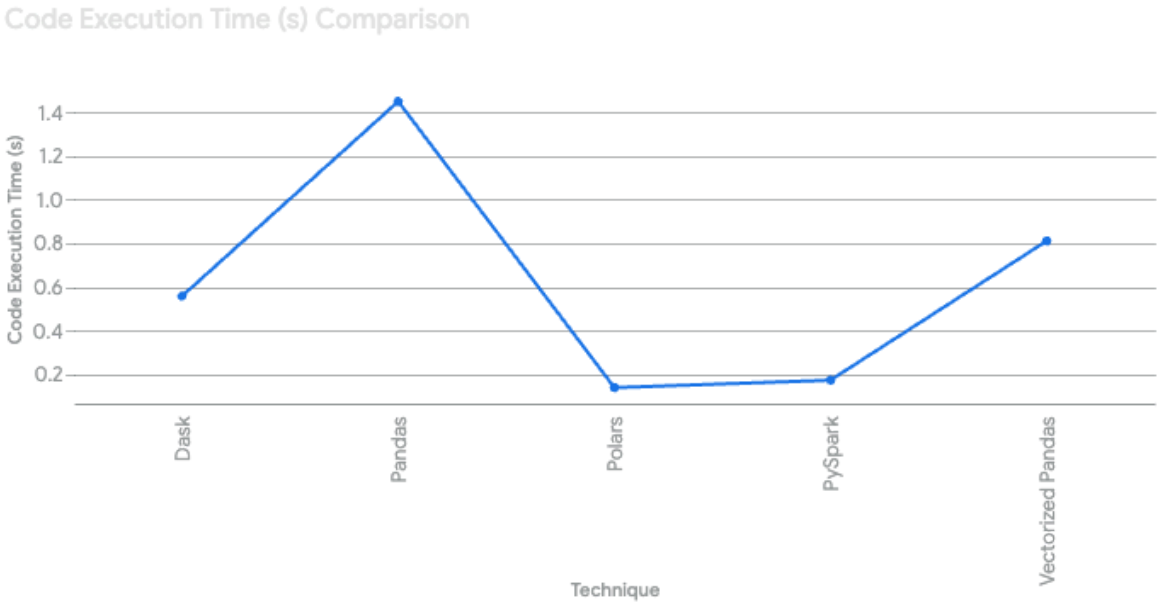


Figure 6.3.3: shows Code Execution Time(s) of each optimization

This third line graph, "Code Execution Time (s) Comparison," assesses the execution time of the same set of methods and is included in the same document. The X-axis once more lists the techniques: Dask, Pandas, Polars, PySpark, and a combination of Vectorize Pandas. The Y-axis displays "Code Execution Time (s)," with values ranging from 0.2 to 1.4 seconds. The graph shows that while the "Polars" and "PySpark" techniques have the shortest execution times, falling well below 0.4 seconds, and "Dask" and the final combined technique fall somewhere in the middle, the "Pandas" technique has the longest execution time, peaking at about 1.4 seconds.

7. Challenges & Limitations

The development of the web crawler was faced with some challenges that required the team to stray from the original plan. Initially, each member of the team was assigned a distinctive web scraping library—such as Scrapy, BeautifulSoup, and Selenium—with the intention of testing their performance and balancing the workload. Unfortunately, this approach did not work during the testing process. Scrapy and BeautifulSoup were not able to handle dynamic page content, pagination, and parts of the New Straits Times (NST) website that were JavaScript-rendered. Because of this, the entire team collectively decided to switch to Selenium, which provided greater more powerful interaction with the browser interface and higher success rates when attempting to scrape the required data. While Selenium did work, it imposed new limitations: it was slower since it was using a full browser render, and it was more resource-intensive. Furthermore, to respect ethical web scraping practices and not infringe on the terms of service of the site, the crawler was intentionally limited to a single user at a time with enforced pauses between requests. To maintain the integrity of the NST site,the process of data collection, making it less scalable for very large datasets or real-time applications.

In the phase of data processing, the team has encountered several technical problems that the performance and efficiency of the task are impacted. The scraped data is scraped in different formats especially in the 'Date' field, in which more than a single stage of string manipulation and type casting are needed in order to standardize. Initial attempts using Pandas were straightforward and easy but required more time as the dataset approached 100,000 rows. In order to optimize the performance, the team used other libraries like Polars, Dask, PySpark and Vectorized Pandas. The libraries provided outstanding speedup in the process and memory gains, but new difficulties also happened. For instance, Dask, as capable as it was in parallel computing, showed high memory usage in certain operations, which is possibly due to internal task graph overheads. PySpark which is good at distributed calculations had slow startup times and wasn't as ideal for smaller jobs. Polars was the quickest and memory-efficient solution, but the syntax was not familiar to the team and required additional time to learn. The second limitation was that the performance benchmarks were executed as single-run tests, instead of having multiple repetitions to factor in system load or execution time fluctuations. This limits the statistical significance of the performance metrics that were reported.

Moreover, the entire pipeline from scraping to cleaning and optimization was following the structure and content of the NST website. Thus, the solution will be least likely to be transferred to websites with different structures, irregular formatting, or anti-scraping features like CAPTCHA or dynamic loading. Future implementations would thus have to consider more flexible scraping mechanisms and modular data processing pipelines. Lastly, the current system works well in a controlled, academic environment, causing it will lack an easily accessible interface for non-technical users to select the most optimal measure of processing based on data size, hardware limitations, or required performance.

8. Conclusion & Future Work

This study compared the performance of unoptimized pandas with optimized data processing methods (Polars, Dask, PySpark, and vectorized pandas) for cleaning and analyzing a dataset loaded from MongoDB. The results demonstrated that Polars outperformed other methods in speed and memory efficiency, achieving ~884K rows/s due to its multithreading, SIMD optimizations, and lazy execution. PySpark also performed well for large-scale processing, while Dask showed higher memory usage. The findings highlight that optimized libraries like Polars can significantly accelerate data workflows without requiring distributed computing.

8.1 Summary of findings

In conclusion, Polars library is the most suitable data processing tool for New Straits Times dataset. From the time comparison graph, Polars only used 0.14 seconds to process all data cleaning operations, which is faster than the other libraries. If compared to pandas, Polars can achieve more than 30x performance gains. This can be seen through the throughput 884573.64 rows per second compared to the 83728.77 rows per second for pandas. Moreover, memory usage for Polars is only 0.38 MB. Therefore, Polars's overall performance is the most efficient and is considered high performance with the help of multiprocessing and multithreading methods.

8.2 What could be improved

The data transformation for Polars can be performed in the future to test the processing speed. The conversion of data type is a challenging task because the initial dataset type is a string data type. As a result, the string data type is not suitable for future analysis and is hard to transform into actionable insight or data-driven insight. Besides that, the performance metrics are based on a single run. Future work could include multiple runs to account for variability and provide more reliable averages. More complex transformations or larger datasets can be implemented to better stress-test the methods. Not only that, future work can include an investigation into why Dask's memory usage was higher and if it can be optimized further. Extend the analysis to include data from other sources, such as SQL databases, to evaluate performance in different contexts. A user-friendly interface or script can be developed to allow users to select the best processing method based on their dataset size and hardware constraints.

9. References

I. Turner-Trauring, "Pandas vectorization: faster code, slower code, bloated memory," Python⇒Speed, Jan. 06, 2023. https://pythonspeed.com/articles/pandas-vectorization/

"Polars vs. pandas: What's the Difference? | The PyCharm Blog," The JetBrains Blog. https://blog.jetbrains.com/pycharm/2024/07/polars-vs-pandas

OpenAI. (2022). *ChatGPT* (Feb 13 version) [Large language model]. https://chat.openai.com

10. Appendices

10.1 Sample code snippets

```
!pip install polars>=0.20.0
!pip install "pymongo[srv]>=4.6.0"
!pip install matplotlib>=3.8.0
!pip install seaborn>=0.13.0
!pip install psutil>=5.9.0
!pip install numpy>=1.24.0


import time
import psutil
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import  regex as re
from bson import ObjectId
from datetime import datetime
from multiprocessing import Pool, cpu_count
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
# polars
import polars as pl
from concurrent.futures import ProcessPoolExecutor, ThreadPoolExecutor

#pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import when, col, regexp_replace, to_date,
trim, initcap

#dask
import dask.dataframe as dd

#pandas
import pandas as pd
```

Figure 1: Code for Installations and Imports

Required libraries:

Polars: High-performance DataFrame library (Rust-based).

PyMongo: MongoDB Python driver with SRV support.

Matplotlib & Seaborn: Data visualization.

psutil: System monitoring (CPU/memory usage).

NumPy: Numerical computing.

The code sets up an environment for benchmarking data processing across multiple libraries (Pandas, Polars, Dask, PySpark). This includes tools for performance tracking (time, psutil) and visualization (matplotlib, seaborn). Last but not least, the code uses pymongo to connect to MongoDB for data loading/saving.

```
def connect_mongodb():
    """Connect to MongoDB and return the client."""
    uri =
"mongodb+srv://josephyeo:fPya67QIXrl4ZsV5@cluster0.ihjgjas.mongodb.net/
?retryWrites=true&w=majority&appName=Cluster0"
    client = MongoClient(uri, server_api=ServerApi('1'))
    try:
        client.admin.command('ping')
        print("Successfully connected to MongoDB!")
        return client
    except Exception as e:
        print(f"Error connecting to MongoDB: {e}")
        return None

# Connect to MongoDB and load data
client = connect_mongodb()
db = client["mydatabase"]
```

Figure 2: Code for MongoDB Connection

connect_mongodb(): This function is to connect to MongoDB and return to the client.

```
file_path =
'https://raw.githubusercontent.com/Jingyong14/HPDP02/refs/heads/main/24
25/project/p1/Group%207/data/raw_data.csv'

# Read the CSV
df_news = pd.read_csv(file_path)
df_news.head()
```

```
row_count = len(df_news)
print("Total number of row:", row_count)

# Create or switch to your database
db = client["mydatabase"]
news_collection = db["raw_news"]

# Delete existing data in the collection
news_collection.delete_many({})

# Insert the all rows into MongoDB
news_collection.insert_many(df_news.to_dict("records"))

print("All rows of news data inserted into MongoDB successfully.")
```

Figure 3: Code for Loading Data

This code block is to load data from csv file into MongoDB.

```
def track_performance(method_name, start_time, start_memory, df):
    """Track performance metrics for data cleaning operations."""
    end_time = time.time()
    end_memory = psutil.Process().memory_info().rss / (1024 * 1024)  #
MB

    time_taken = end_time - start_time
    throughput = len(df) / time_taken if time_taken > 0 else 0
    memory_used = end_memory - start_memory  # in MB

    return {
        "Method": method_name,
        "Time (s)": time_taken,
        "Throughput (rows/s)": throughput,
        "Memory Used (MB)": memory_used
    }

def track_performance_pyspark(method_name, start_time, start_memory,
df=None):
    end_time = time.time()
    end_memory = psutil.Process().memory_info().rss / (1024 * 1024)

    time_taken = end_time - start_time

    if df is not None:
```

```
        if hasattr(df, "count"):  # PySpark DataFrame
            row_count = df.count()
        else:  # pandas, dask, polars
            row_count = len(df)
    else:
        row_count = 0

    throughput = row_count / time_taken if time_taken > 0 else 0
    memory_used = end_memory - start_memory  # in MB

    return {
        "Method": method_name,
        "Time (s)": round(time_taken, 4),
        "Throughput (rows/s)": round(throughput, 2),
        "Memory Used (MB)": round(memory_used, 2)
    }
```

Figure 4: Performance Tracking Function Code

The track_performance function calculates the time taken, memory usage, and throughput for a given data processing operation.

- It takes four arguments:
  - method_name: A string representing the name of the data processing method being tracked (e.g., "Pandas", "Polars").
  - start_time: The time recorded at the beginning of the operation using time.time().
  - start_memory: The memory usage recorded at the beginning of the operation using psutil.Process().memory_info().rss.
  - df: The DataFrame object after the operation is completed.
- Inside the function, it records the end_time and end_memory. Memory is converted from bytes to megabytes.
- It calculates time_taken by subtracting start_time from end_time.
- Throughput is calculated as the number of rows processed (len(df)) divided by the time_taken. A check is included to prevent division by zero.
- memory_used is calculated as the difference between end_memory and start_memory.
- Finally, it returns a dictionary containing the Method name and the calculated performance metrics: Time (s), Throughput (rows/s), and Memory Used (MB).

```
# Load documents into a DataFrame
db = client["mydatabase"]
collection = db['raw_news']
data = list(collection.find())
df_panda = pd.DataFrame(data)

def clean_data(df):
    start_time = time.time()
```

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

    df = df.drop(columns=['_id'])

    # Drop nulls and duplicates
    df = df.dropna()
    df = df.drop_duplicates(keep='first')

    # Standardize 'Section' column
    if 'Section' in df.columns:
        df["Section"] = df["Section"].str.title()

    # Clean 'Date' column
    def clean_date(date):
        if isinstance(date, str):
            if "@" in date:
                date = date.split("@")[0].strip()
            return re.sub(r'\s+', ' ', date)
        return date

    if 'Date' in df.columns:
        df["Date"] = df["Date"].apply(clean_date)
        df["Date"] = pd.to_datetime(df["Date"], errors='coerce')  #
Invalid dates become NaT


    # Track performance using shared function
    performance_report = track_performance("Pandas ", start_time,
start_memory, df)

    return df, performance_report


cleaned_df, pandas_result = clean_data(df_panda)

print("Final row: ", len(cleaned_df))
# Save cleaned data to MongoDB
cleaned_data = cleaned_df.to_dict("records")
db["cleaned_data_pandas"].delete_many({})
db["cleaned_data_pandas"].insert_many(cleaned_data)

print("Cleaned data inserted into 'cleaned_data_pandas' collection in
MongoDB")
```

```
print("")
print("Pandas cleaning completed!")
print(pandas_result)
print("")

# Save cleaned data to CSV
cleaned_df.to_csv("cleaned_data_unoptimized_pandas.csv", index=False)
print("Cleaned data saved to 'cleaned_data_unoptimized_pandas.csv'")
```

Figure 5: Pandas Data Processing Code

clean_data(df) : This function cleans a Pandas DataFrame using optimized operations and tracks performance.

The figure 5 also shows the coding step for performance tracking set up, drop mongoDB _id column, data cleaning processes such as drop nulls/duplicates, standardize column and clean date format, run cleaning, return result, save result to mongoDB and to csv and print summary.

```
def load_data(client):
    """Load data from MongoDB into a Polars DataFrame."""
    db = client["mydatabase"]
    collection = db['raw_news']
    data = list(collection.find())
    df = pl.DataFrame(data)
    return df

df_polars = load_data(client)

def clean_data_polars_default(df):
    """Clean data using Polars' default processing."""
    start_time = time.time()
    start_memory = psutil.Process().memory_info().rss / (1024 * 1024)

    df = df.drop('_id')

    # Enable string cache for better performance
    with pl.StringCache():

        # Define list of fake nulls
        fake_nulls = ["", "NaN", "null"]

        # Apply replacement for all columns
        df = df.with_columns([
            pl.when(pl.col(col).is_in(fake_nulls))
              .then(None)
              .otherwise(pl.col(col))
              .alias(col)
            for col in df.columns
```

```
        ])
        # Drop duplicates and nulls
        df = df.drop_nulls()
        df = df.unique()
        # Standardize Section
        if 'Section' in df.columns:
            df = df.with_columns(
                df['Section'].str.to_titlecase().alias('Section')
            )

        # Clean Date
        if 'Date' in df.columns:
            df = df.with_columns(
                df['Date']
                .str.split('@')
                .list.first()
                .str.strip_chars()
                .alias('Date')
            )

            # Convert to datetime
            df = df.with_columns(
               pl.col('Date').str.strptime(pl.Datetime, format='%b %d,
%Y', strict=False).alias('Date')
            )
            df = df.filter(pl.col("Date").is_not_null())

    return df, track_performance("Polars", start_time, start_memory,
df)

# Run default Polars processing
df_polar_cleaned, polar_result = clean_data_polars_default(df_polars)

print("Final row: ", len(df_polar_cleaned.to_pandas()))
# Save cleaned data to MongoDB
polar_cleaned = df_polar_cleaned.to_dicts()
db["cleaned_data_polars"].delete_many({})
db["cleaned_data_polars"].insert_many(polar_cleaned)
print("Cleaned data inserted into 'cleaned_data_polars' collection in
MongoDB")
print("")
```

```
df_polar_cleaned =
df_polar_cleaned.to_pandas().to_csv("cleaned_data_optimized_polar.csv",
index=False)
print("Cleaned data saved to 'cleaned_data_optimized_polar.csv'")
print("")

print("Polars cleaning completed!")
print(polar_result)
```

Figure 6: Polars Data Processing Code

load_data(client): This function load data into Polars dataframe.
clean_data_polars_default(df) : This function cleans a Polars DataFrame using optimized operations and tracks performance.
The figure 6 also shows the coding step for performance tracking set up, drop mongoDB _id column, string caching optimization, handles “Fake Nullls”, data cleaning processes such as drop nulls/duplicates, standardize column and clean date format, run cleaning, return result, save result to mongoDB and to csv and print summary.

```
df_dask = dd.from_pandas(df_panda, npartitions=4)
# Function to clean data using Dask (default scheduler)
def clean_data_dask_default(df_dask):
    start_time = time.time()
    start_memory = psutil.Process().memory_info().rss / (1024 * 1024)
# MB

    df_cleaned = df_dask.drop(columns=['_id'])

    df_cleaned = df_cleaned.map_partitions(lambda df: df.dropna())

    # Perform drop_duplicates() in Pandas (to check across all rows),
then switch back to Dask
    df_cleaned_pandas = df_cleaned.compute()
    df_cleaned_pandas =
df_cleaned_pandas.drop_duplicates(subset=["Section", "Date",
"Headline", "Summary"])

    # Convert back to Dask DataFrame after dropping duplicates
    df_cleaned = dd.from_pandas(df_cleaned_pandas, npartitions=4)

    df_cleaned = df_cleaned.map_partitions(lambda df:
df.assign(Section=df['Section'].str.title()))
    df_cleaned = df_cleaned.map_partitions(lambda df:
df.assign(Date=df['Date'].map(lambda x: x.split('@')[0].strip() if
isinstance(x, str) else x)))
```

```
    df_cleaned = df_cleaned.map_partitions(lambda df:
df.assign(Date=pd.to_datetime(df['Date'], errors='coerce')))

    # Compute to Pandas
    cleaned_df = df_cleaned.compute()

    # Track performance
    performance_report = track_performance("Dask ", start_time,
start_memory, cleaned_df)

    return cleaned_df, performance_report

df_dask_cleaned, dask_result = clean_data_dask_default(df_dask)

print("Final row: ", len(df_dask_cleaned))
# Save cleaned data to MongoDB
dask_cleaned = df_dask_cleaned.to_dict("records")
db["cleaned_data_dask"].delete_many({})
db["cleaned_data_dask"].insert_many(dask_cleaned)



df_dask_cleaned.to_csv("cleaned_data_optimized_dask.csv", index=False)
print("Cleaned data saved to 'cleaned_data_optimized_dask.csv'")
print("")

print("Cleaned data inserted into 'cleaned_data_dask' collection in
MongoDB")
print("")
print("Dask cleaning completed!")
print(dask_result)
```

Figure 7: Dask Data Processing Code

df_dask = dd.from_pandas(df_panda, npartitions=4): This line is to create a dask dataframe from a pandas dataframe.

df_dask: This is the variable that will hold the newly created Dask DataFrame.

dd.from_pandas(): This is a function from the Dask library (aliased as dd) specifically designed to convert an existing Pandas DataFrame into a Dask DataFrame.

df_panda: This is the input Pandas DataFrame that you want to convert.

npartitions=4: This argument specifies the number of partitions to divide the data into. Dask works by breaking large datasets into smaller chunks (partitions) and processing these partitions in parallel. Setting npartitions=4 means that the original df_panda will be split into 4 smaller dataframes, and these will form the partitions of the df_dask DataFrame. The choice of the number of partitions can impact performance; more partitions might offer better parallelism but can also introduce overhead.

clean_data_dask_default(df): This function cleans a Dask DataFrame using dask schedular and tracks performance.
The figure 7 also shows the coding step for performance tracking set up, data cleaning processes such as drop nulls/duplicates, standardize column and clean date format, return result, run cleaning, save result to mongoDB and to csv and print summary.

```
def clean_data_pyspark(df_spark):
    start_time = time.time()
    start_memory = psutil.Process().memory_info().rss / (1024 * 1024)

    # Drop NA and Duplicates
    df_cleaned= df_spark.drop('_id')

    # List of "fake" nulls to replace
    fake_nulls = ["", "NaN", "null"]

    # Replace in all columns
    for c in df_cleaned.columns:
        df_cleaned = df_cleaned.withColumn(c,
            when(col(c).isin(fake_nulls), None).otherwise(col(c))
        )

    df_cleaned = df_cleaned.dropna()
    df_cleaned = df_cleaned.dropDuplicates()

    # Clean and convert the Date column
    df_cleaned = df_cleaned.withColumn("Date",
regexp_replace(col("Date"), "@.*$", ""))  # Remove time part
    df_cleaned = df_cleaned.withColumn("Date",
regexp_replace(col("Date"), "\s+", " "))  # Normalize spaces
    df_cleaned = df_cleaned.withColumn("Date", trim(col("Date")))  #
Trim spaces
    df_cleaned = df_cleaned.withColumn("Date", to_date(col("Date"),
"MMM d, yyyy"))  # Parse to date
    df_cleaned = df_cleaned.filter(col("Date").isNotNull())


    # Format Section to Title Case
    df_cleaned = df_cleaned.withColumn("Section",
initcap(col("Section")))

    return df_cleaned, track_performance_pyspark("PySpark", start_time,
start_memory, df_cleaned)
```

```
# Initialize Spark
spark = SparkSession.builder.appName("CleanNewsData").getOrCreate()

# Convert MongoDB ObjectId to string
df_panda["_id"] = df_panda["_id"].astype(str)

# Convert to PySpark DataFrame
df_spark = spark.createDataFrame(df_panda)

# Clean and track performance
df_spark_cleaned, pyspark_result = clean_data_pyspark(df_spark)

# Save cleaned data to MongoDB
# Convert PySpark DataFrame to Pandas DataFrame
df_spark_cleaned_pandas = df_spark_cleaned.toPandas()

# Fix dates in pandas df
df_spark_cleaned_pandas =
df_spark_cleaned_pandas[pd.to_datetime(df_spark_cleaned_pandas["Date"],
errors="coerce").notna()]
df_spark_cleaned_pandas["Date"] =
pd.to_datetime(df_spark_cleaned_pandas["Date"])
print("Final row: ", len(df_spark_cleaned_pandas))

# Convert to dictionary for MongoDB insertion
spark_cleaned = df_spark_cleaned_pandas.to_dict("records")
db["cleaned_data_pyspark"].delete_many({})
db["cleaned_data_pyspark"].insert_many(spark_cleaned)
print("Cleaned data inserted into 'cleaned_data_pyspark' collection in
MongoDB\n")

# Save to CSV
df_spark_cleaned_pandas.to_csv("cleaned_data_optimized_pyspark.csv",
index=False)
print("Cleaned data saved to 'cleaned_data_optimized_pyspark.csv'\n")

print("Pyspark cleaning completed!")
print(pyspark_result)
```

Figure 8: PySpark Data Processing Code

clean_data_pyspark(df): This function cleans a Polars DataFrame using optimized operations and tracks performance.

The figure 8 also shows the coding step for performance tracking set up, drop mongoDB _id column, handles “Fake Nullls”, data cleaning processes such as drop nulls/duplicates, standardize column and

clean date format, return result, initialize spark, convert mongoDB Objectid to string, convert to PySpark Dataframe, run cleaning, save result to mongoDB and to csv and print summary.

```
# Optimized vectorized cleaning function
def clean_data_pandas_vectorized(df):
    start_time = time.time()
    start_memory = psutil.Process().memory_info().rss / (1024 * 1024)

    # Drop nulls and duplicates in one go (vectorized)
    df = df.drop(columns=['_id'])
    df = df.dropna().drop_duplicates()

    # Standardize 'Section' column (vectorized str methods)
    if 'Section' in df.columns:
        df['Section'] = df['Section'].str.title()

    # Clean 'Date' column (vectorized string methods + datetime)
    if 'Date' in df.columns:
        # Clean strings using vectorized apply (no need for a loop)
        df['Date'] = df['Date'].str.split('@').str[0].str.strip()
        df['Date'] = df['Date'].str.replace(r'\s+', ' ', regex=True)

        # Convert to datetime in one call (vectorized)
        df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

    # Track performance (reuse track_performance function)
    performance_report = track_performance("Vectorized Pandas",
start_time, start_memory, df)

    return df, performance_report

# Test the function
df_vectorized_cleaned, vectorized_result =
clean_data_pandas_vectorized(df_panda)

# Save cleaned data to MongoDB
vectorized_cleaned = df_vectorized_cleaned.to_dict("records")
db["cleaned_data_vectorized_pandas"].delete_many({})
db["cleaned_data_vectorized_pandas"].insert_many(vectorized_cleaned)
print("Cleaned data inserted into 'cleaned_data_vectorized_pandas'
collection in MongoDB")
print("")
```

```
df_vectorized_cleaned.to_csv("cleaned_data_optimized_vectorized.csv",
index=False)
print("Cleaned data saved to 'cleaned_data_optimized_vectorized.csv'")
print("")
print("Vectorized pandas cleaning completed!")
print(vectorized_result)
```

Figure 9: Vectorized Pandas Data Processing Code

clean_data_pandas_vectorized(df): This function cleans a Polars DataFrame using optimized operations and tracks performance.

The figure 9 also shows the coding step for performance tracking set up, drop mongoDB _id column, data cleaning processes such as drop nulls/duplicates, standardize column and clean date format using vectorized string methods, return result, run cleaning, save result to mongoDB and to csv and print summary.

```
def plot_results(results):
    """Plot performance comparison results in 3 columns"""
    # Convert results to pandas DataFrame for plotting
    results_df = pl.DataFrame(results).to_pandas()

    # Set plot style
    sns.set(style="whitegrid")

    # Define metrics and color palette
    metrics = ["Time (s)", "Throughput (rows/s)", "Memory Used (MB)"]
    palette = sns.color_palette("pastel",
n_colors=len(results_df['Method'].unique()))

    # Create a single row of 3 subplots
    fig, axes = plt.subplots(1, 3, figsize=(12,4))

    for i, metric in enumerate(metrics):
        ax = axes[i]
        sns.barplot(
            x="Method", y=metric, hue="Method", legend=False,
            data=results_df, palette=palette, ax=ax
        )

        # Add values on top of the bars
        for p in ax.patches:
            ax.annotate(f'{p.get_height():.2f}',
                        (p.get_x() + p.get_width() / 2.,
p.get_height()),
                        ha='center', va='center', fontsize=10,
color='black',
```

```
                        xytext=(0, 5), textcoords='offset points')

        ax.set_title(f"{metric} Comparison", fontsize=14)
        ax.set_xlabel("")
        ax.set_ylabel(metric)

    plt.tight_layout()
    plt.show()


# Collect all results
results = [polar_result, pandas_result]

# Plot results
plot_results(results)
```

Figure 10: Plot Result Code

This code block defines a function called plot_results and then uses it to visualize the performance comparison between two data processing methods.

Breakdown of what plot_results(results) function does:

- Convert Results to Dataframe
- Set Plot Style
- Define Metrics and Color Palette
- Create Subplots
- Iterate and Plot Each Metrice
- Annotate Bar
- Set Plot Titles and Labels
- Adjust Layout and Show Plot

```
def plot_results_overall(results):
    """Plot performance comparison results in 3 columns
(side-by-side)."""
    import matplotlib.pyplot as plt
    import seaborn as sns
    import polars as pl

    # Convert results to pandas DataFrame
    results_df = pl.DataFrame(results).to_pandas()

    # Set seaborn style
    sns.set(style="whitegrid")

    # Define metrics and color palette (5 distinct pastel colors)
    metrics = ["Time (s)", "Throughput (rows/s)", "Memory Used (MB)"]
```

```
    unique_methods = results_df['Method'].unique()
    palette = dict(zip(unique_methods, sns.color_palette("pastel",
n_colors=len(unique_methods))))

    # Create subplots
    fig, axes = plt.subplots(1, 3, figsize=(16, 5))

    for i, metric in enumerate(metrics):
        ax = axes[i]
        sns.barplot(
            x="Method", y=metric, hue="Method",
            data=results_df, palette=palette, legend=False, ax=ax
        )

        # Annotate bar values
        for p in ax.patches:
            height = p.get_height()
            ax.annotate(f'{height:.2f}',
                        (p.get_x() + p.get_width() / 2., height),
                        ha='center', va='bottom', fontsize=9,
color='black',
                        xytext=(0, 5), textcoords='offset points')

        ax.set_title(f"{metric} Comparison", fontsize=13)
        ax.set_xlabel("")
        ax.set_ylabel(metric)
        ax.tick_params(axis='x', labelrotation=45)

    plt.tight_layout()
    plt.show()


# Collect all results
overall = [pandas_result, vectorized_result, polar_result, dask_result,
pyspark_result ]

# Plot results
plot_results_overall(overall)
```

Figure 11: All Comparison Code Among Libraries

```
# Close the MongoDB connection
client.close()
```

Figure 12: Clean Up Code

10.2 Screenshots of output

```
Requirement already satisfied: pymongo>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from pymongo[srv]>=4.6.0) (4.12.1)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from pymongo>=4.6.0->pymongo[srv]>=4.6.0) (2.7.0)
WARNING: pymongo 4.12.1 does not provide the extra 'srv'
```

Output 1: Installations complete

```
Successfully connected to MongoDB!
```

Output 2: Connected to MongoDB

```
Total number of row: 127729
All rows of news data inserted into MongoDB successfully.
```

Output 3: Data insertion into MongoDB

```
Final row:  121437
Cleaned data inserted into 'cleaned_data_pandas' collection in MongoDB

Pandas cleaning completed!
{'Method': 'Pandas ', 'Time (s)': 1.450361728668213, 'Throughput (rows/s)': 83728.76752029915, 'Memory Used (MB)': 0.0}

Cleaned data saved to 'cleaned_data_unoptimized_pandas.csv'
```

Output 4: Pandas Data Processing

```
Final row:  121437
Cleaned data inserted into 'cleaned_data_polars' collection in MongoDB

Cleaned data saved to 'cleaned_data_optimized_polar.csv'

Polars cleaning completed!
{'Method': 'Polars', 'Time (s)': 0.1372830867767334, 'Throughput (rows/s)': 884573.6415986606, 'Memory Used (MB)': 0.3828125}
```

Output 5: Polars Data Processing

```
<ipython-input-143-fedb2e56790b>:19: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and a
  df_cleaned = df_cleaned.map_partitions(lambda df: df.assign(Date=pd.to_datetime(df['Date'], errors='coerce')))
Final row:  121437
Cleaned data saved to 'cleaned_data_optimized_dask.csv'

Cleaned data inserted into 'cleaned_data_dask' collection in MongoDB

Dask cleaning completed!
{'Method': 'Dask ', 'Time (s)': 0.5574398040771484, 'Throughput (rows/s)': 217847.7373015031, 'Memory Used (MB)': 9.56640625}
```

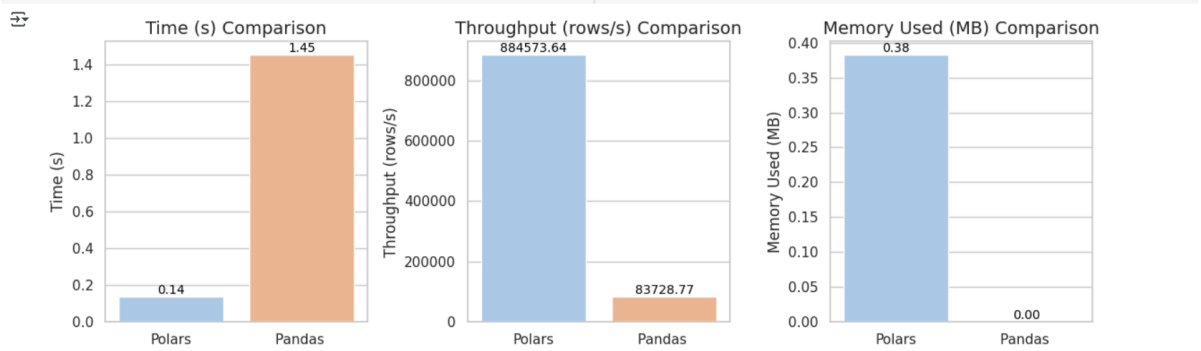
Output 6: Dask Data Processing

```
Final row:  121437
Cleaned data inserted into 'cleaned_data_pyspark' collection in MongoDB

Cleaned data saved to 'cleaned_data_optimized_pyspark.csv'

Pyspark cleaning completed!
{'Method': 'PySpark', 'Time (s)': 0.1715, 'Throughput (rows/s)': 707961.62, 'Memory Used (MB)': 0.0}
```

Output 7: PySpark Data Processing

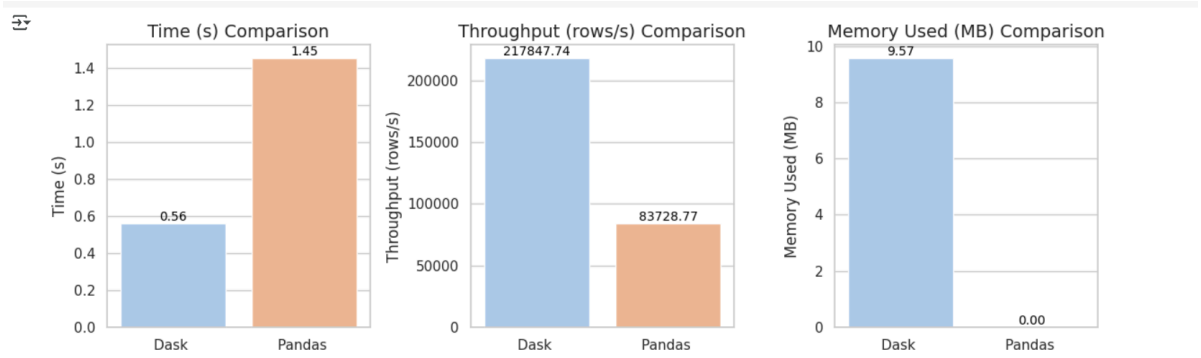
```
Cleaned data inserted into 'cleaned_data_vectorized_pandas' collection in MongoDB

Cleaned data saved to 'cleaned_data_optimized_vectorized.csv'

Vectorized pandas cleaning completed!
{'Method': 'Vectorized Pandas', 'Time (s)': 0.810391902923584, 'Throughput (rows/s)': 149849.72031667858, 'Memory Used (MB)': 0.0}
```

Output 8: Vectorized Pandas Data Processing



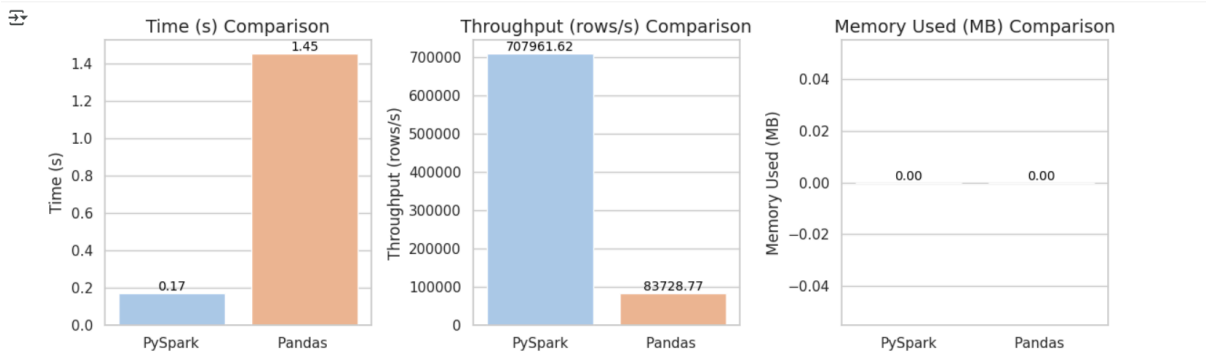
Output 9: Graph of Polars vs Pandas

Polars (Optimized): Significantly faster with a throughput of ~884,573 rows/s and used 0.38 MB of additional memory. The cleaning process took ~0.137 seconds.



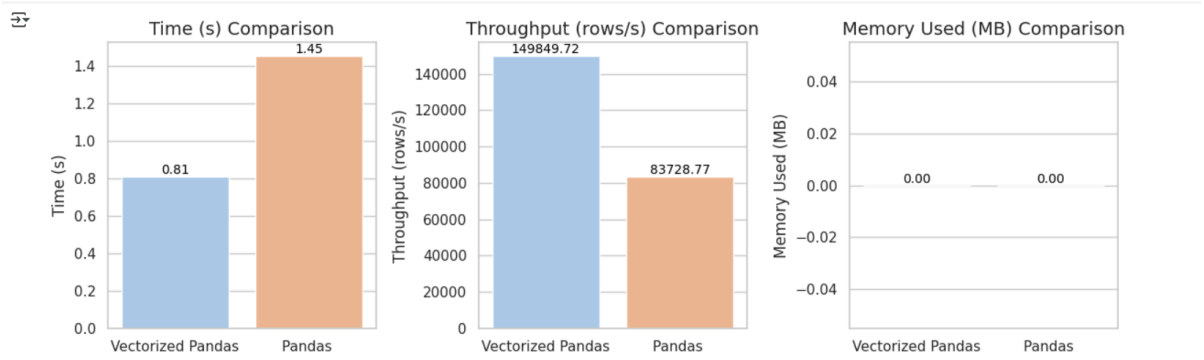
Output 10: Graph of Dask vs Pandas

Dask (Optimized): Achieved a throughput of ~217,847 rows/s and used 9.57 MB of additional memory. The cleaning process took ~0.557 seconds.



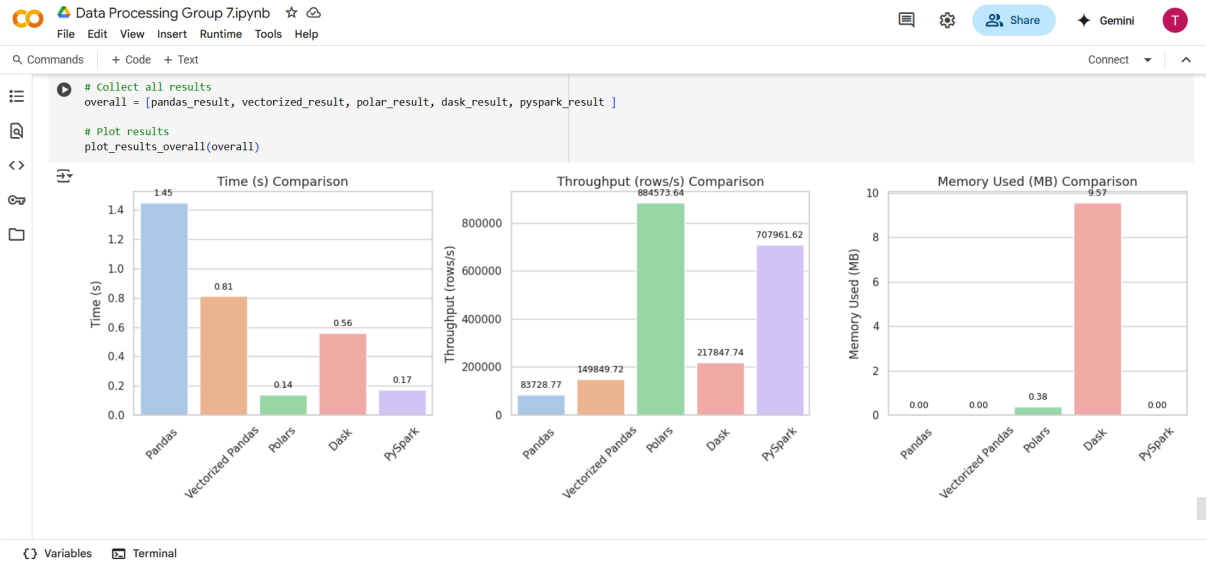
Output 11: Graph of PySpark and Pandas

PySpark (Optimized): Achieved a throughput of ~707,961 rows/s and used 0 MB of additional memory. The cleaning process took ~0.171 seconds.



Output 12: Graph of Vectorized Pandas vs Pandas

Vectorized Pandas (Optimized): Achieved a throughput of ~149,849 rows/s and used 0 MB of additional memory. The cleaning process took ~0.81 seconds.



Output 13: All Graphs Compared with Pandas

Pandas (Unoptimized): Achieved a throughput of ~83,728 rows/s and used 0 MB of additional memory. The cleaning process took ~1.45 seconds.

10.3 Links to full code repo or dataset

Raw Dataset link:
https://github.com/Jingyong14/HPDP02/blob/main/2425/project/p1/Group%207/data/raw_data.xlsx
Data Processing File Link:
https://colab.research.google.com/drive/1khMiYXUq926IHhzo20M8q18WZEOx_QCy?usp=sharing#scrollTo=w1RDQKQkfOYt