



**UTM**  
UNIVERSITI TEKNOLOGI MALAYSIA

**SECP3133 HIGH-PERFORMANCE DATA PROCESSING**

**SEMESTER 2 2024/2025**

**OPTIMIZING HIGH-PERFORMANCE DATA PROCESSING  
FOR LARGE-SCALE WEB CRAWLERS REPORT**

**SECTION: 01**

**LECTURER: ASSOC. PROF. DR. MOHD SHAHIZAN BIN  
OTHMAN**

**GROUP NAME: GROUP E**

<b>NO.</b>	<b>NAME</b>	<b>MATRIC NO.</b>
1	DANIAL HARRIZ BIN MOHD ASINEH @MOHD ASNEH	A22EC0152
2	CHAI YU TONG	A22EC0145
3	KOH SU XUAN	A22EC0060
4	TIEW CHUAN RONG	A22EC0112

**SUBMISSION DATE :**

**16<sup>th</sup> MAY 2025**

# Table of Contents

<b>1.0 Introduction.....</b>	<b>1</b>
1.1. Background of the project.....	1
1.2. Objectives.....	1
1.3. Target website and data to be extracted.....	2
<b>2.0 System Design &amp; Architecture.....</b>	<b>3</b>
2.1. Description of architecture.....	3
2.2. Tools and frameworks used.....	4
2.3. Roles of team members.....	5
<b>3.0 Data Collection.....</b>	<b>6</b>
3.1. Crawling method.....	6
3.2. Number of records collected.....	6
3.3. Ethical considerations.....	7
<b>4.0 Data Processing.....</b>	<b>8</b>
4.1. Cleaning methods.....	8
4.2. Data structure.....	8
4.3. Transformation and formatting.....	8
<b>5.0 Optimization Techniques.....</b>	<b>9</b>
5.1. Methods used.....	9
5.2. Code overview or pseudocode of techniques applied.....	10
<b>6.0 Performance Evaluation.....</b>	<b>18</b>
6.1. Before vs after optimization.....	18
6.2. Processing time, CPU usage, memory usage and throughput.....	18
6.3. Charts and Graphs.....	24
<b>7.0 Challenges &amp; Limitations.....</b>	<b>28</b>
<b>8.0 Conclusion &amp; Future Work.....</b>	<b>29</b>
8.1. Summary of findings.....	29

8.2. What could be improved.....	30
<b>References.....</b>	<b>31</b>
<b>Appendices.....</b>	<b>32</b>
Sample Code Snippets.....	32
Screenshots of output.....	36
Links to full code repo or dataset.....	39

## **1.0 Introduction**

### **1.1. Background of the project**

Online news platforms have become an important source of information on a wide range of topics which includes politics, economics, social issues, and also events that are happening around the world. With the increasing availability of public data on news websites, web scraping is a powerful tool to automate the collection of large volumes of content for different purposes such as sentiment analysis, trend detection, and content archiving.

This project uses web scraping techniques to extract news articles from an established Malaysian online news portal by leveraging a high-performance scraping pipeline. The end goal is to gather relevant news data for further processing and performance analysis.

### **1.2. Objectives**

The main objectives of this project are:

- To develop a web scraper that is capable of collecting up to 100,000 news article records from a target website.
- Extracting specific data elements such as article title, URL, teaser, and category.
- To store the extracted data in a structured CSV format that will be used for tasks such as processing and performance analysis.
- Clean the data collected from web scraping.
- Compare the three libraries (Pandas, Polars and Modin) for data processing.
- Optimize data processing methods to get better performance.

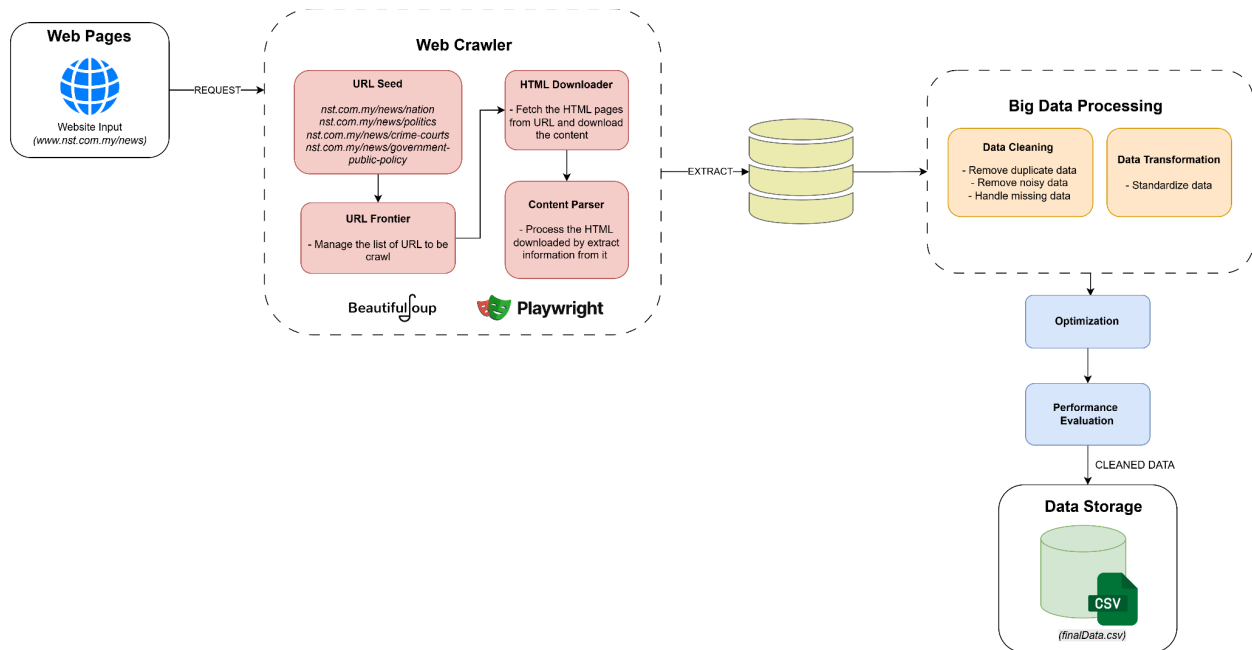
### 1.3. Target website and data to be extracted

The website that is targeted for this web scraping project is the New Straits Times (NST) under the “News” section. It is accessible via <https://www.nst.com.my/news/>. This section displays updated news articles that are related to the national developments in Malaysia.

The specific data fields that will be extracted from each article are:

- **Title:** The headline/main title of the article.
- **URL:** The full web link/URL of the article.
- **Teaser:** A short description from the article content.
- **Category:** The label that describes the main topic area of the article.

## 2.0 System Design & Architecture



**Figure 2.0.1: System Architecture**

### 2.1. Description of architecture

This system architecture illustrates the workflow of our web scraping and data preprocessing pipeline that extracts, cleans and stores structured news data from the New Straits Times (<https://www.nst.com.my/news>). This system follows a Data Flow Architecture Pattern where data flows through a linear sequence of processing stages of Input, Intermediate Processing and Output. The architecture consists of:

- **Web Pages**

The system begins with a crawl request for URLs under 4 categories, i.e. nation, politics, crime-courts and government-public-policy. These URLs serve as the seed input to the crawler.

- **Web Crawler**

We use BeautifulSoup and Playwright to build the crawler. There is a URL seed, URL frontier to manage and maintain the list of URLs to be crawled.

The HTML downloader is to fetch and download HTML pages from the URLs and the content parser is to parse the downloaded HTML to extract the target data field.

- Raw Data Storage

The content that has been parsed is saved as raw data in a CSV file to be processed afterward.

- Big Data Processing

The content that has been parsed is saved as raw data in a CSV file to be processed afterward.

- Optimization & Performance Evaluation

This project uses Pandas as a benchmarking library and optimization libraries which are Polars and Modin to evaluate the efficiency of data processing in large-scale settings.

- Data Storage

The cleaned and processed data is saved in a structured CSV format.

## **2.2. Tools and frameworks used**

- BeautifulSoup
- Playwright
- Asyncio
- Pandas
- Polars
- Modin

### 2.3. Roles of team members

Team Member	Roles
DANIAL HARRIZ BIN MOHD ASINEH @MOHD ASNEH	Group Leader, Coder
TIEW CHUAN RONG	Architect, Coder
CHAI YU TONG	Data Analyst, Documentation Lead, Coder
KOH SU XUAN	HPC Specialist, Evaluator, Coder



## 3.0 Data Collection

### 3.1. Crawling method

The collection of data was performed using a custom-built asynchronous web scraper that was developed using Playwright, which is a browser automation library. The scraper navigates through the pages of the NST News in each subcategory section, where each page lists 20 news articles.

The strategies employed to ensure efficient and robust data collection are:

- **Pagination:** URLs were generated in the format of `https://www.nst.com.my/news/{subcategory}?page={page_number}`.
- **Asynchronous Processing:** The scraper utilizes asynchronous processing with Playwright and `asyncio`, where up to 8 concurrent pages are run to handle multiple pages in parallel, improving efficiency.
- **Rate Limiting and Random Delays:** To reduce server load and mimic human behavior, random delays are set between requests. In addition, a retry mechanism was implemented to ensure robustness.

### 3.2. Number of records collected

The number of records collected was **110,642** across four distinct subcategories under the News section of the New Straits Times:

- Crime & Courts
- Nation
- Government/Public Policy
- Politics

Each subcategory was crawled independently, where each of the members in our group was responsible for scraping one or two subcategories. The same data fields are scraped to ensure that the data collected are uniform and can be merged.

### 3.3. Ethical considerations

The web scraper was developed with adherence to the ethical guidelines to ensure responsible data collection was done:

- **Respect site rules:** Only publicly accessible content was accessed. This is to ensure that the scraping did not violate the website's terms of service (TOS).
- **Human-like interaction:** Rate-limiting and random delays implementation mimics human browsing patterns which reduces the risk of server overload.
- **Non-commercial use:** The data collected will only be used for academic and research purposes. No redistribution of the scraped content is intended.

## 4.0 Data Processing

### 4.1. Cleaning methods

#### a. Handle Duplicate Data

Some of the news might have more than one tag, hence we might scrap the same news under different categories. To solve this, any duplicate rows from the dataset are removed to avoid redundant data.

```
df_cleaned = rd.drop_duplicates()
```

#### b. Handle Missing Data

Some rows may have missing values in the key column (Teaser). Removed using dropna() to maintain data quality.

```
df_cleaned = df_cleaned.dropna()
```

### 4.2. Data structure

The initial data is read from a Microsoft Excel file.

```
rd = pd.read_excel("NST_News_Articles.xlsx")
```

The final data is exported in CSV format.

```
sorted_df.to_csv('finalData.csv', index=False)
```

### 4.3. Transformation and formatting

In the place column, it has city names.

- Converts all text to uppercase. Standardize the format so “kuala lumpur” and “KUALA LUMPUR” are treated the same.
- Splits the value on the ; to keep only the first part.
- Removes any non-alphabetic characters to ensure the values only contain clean, readable city names.




```
df_cleaned['Place'] = df_cleaned['Place'].str.upper()  
df_cleaned['Place'] = df_cleaned['Place'].str.split(';').str[0]  
df_cleaned['Place'] = df_cleaned['Place'].str.replace(r'[^a-zA-Z\s]+', '', regex=True)
```

## 5.0 Optimization Techniques

### 5.1. Methods used

In this project, the Pandas library is used for traditional data processing without any optimization. Pandas is a widely used tool for data processing in Python. On the other hand, Polars and Modin are modern libraries that are selected to optimize the process of data processing. Both these libraries have the capabilities to use multithreading and multiprocessing to distribute tasks into smaller tasks and run them in multiple CPU cores. In this part, we would like to compare these three libraries on how they perform optimization techniques in handling large datasets. Below are the introductions of the three libraries that we used to process data:

*Table 5.1.1: Libraries Used to Process Data and Explanations*

Library	Explanation
 <b>Pandas</b>	<ul style="list-style-type: none"><li>• Pandas is a traditional, simple and easy language that is used by many people to handle data in python.</li><li>• Good for small and medium sized dataset</li><li>• Pandas does not support parallel processing, it uses only one CPU core to process that data.</li><li>• Pandas run in a single process and on a single thread only.</li></ul>
 <b>Polars</b>	<ul style="list-style-type: none"><li>• Polars is a modern tool that is built with Rust.</li><li>• Best for big data that needs optimization and performance.</li><li>• Polars use parallel processing, it uses all CPU cores in the computer to process a lot of data at same time.</li><li>• Polars uses multiprocessing and multithreading to split tasks and run them in parallel.</li></ul>
 <b>Modin</b>	<ul style="list-style-type: none"><li>• Modin can speed up workflows by scaling pandas.</li><li>• Modin works well on larger datasets.</li><li>• Modin uses parallel processing, it uses multiple CPU cores in the computer to handle the task.</li><li>• Modin speeds up pandas work without rewriting the whole code.</li><li>• Modin uses Ray or Dask, which break work into chunks and run them across the CPU core for multiprocessing.</li></ul>

## 5.2. Code overview or pseudocode of techniques applied

### 1. Import library

#### a. Pandas

- Import the Pandas library

```
[3] import pandas as pd
import re
import time
import psutil
```

#### b. Polars

- Import the Polars library, a DataFrame library and other libraries to monitor the performance.

```
[ ] import polars as pl
import re
import time
import psutil
```

#### c. Modin

- Import the Modin version of the Pandas library and other libraries to monitor the performance.

```
[16] import modin.pandas as pd
import re
import time
import psutil
```

### 2. Load data

#### a. Pandas

- Load the raw dataset from the NST\_News\_Articles.csv file using pd.read\_excel.

```
[5] rd = pd.read_excel("NST_News_Articles.xlsx")
```

#### b. Polars

- Load the raw dataset from the NST\_News\_Articles.csv file using pl.read\_excel.

```
[ ] rd = pl.read_excel("NST_News_Articles.xlsx")

print(calculate_processing_metrics(rd))
```

c. Modin

- Load the raw dataset from the NST\_News\_Articles.csv file using pd.read\_excel.

```
[18] rd = pd.read_excel("NST_News_Articles.xlsx")

print(calculate_processing_metrics(rd))
```

### 3. Remove duplicated data

a. Pandas

- Remove duplicate rows from the table using drop\_duplicates()

```
[6] df_cleaned = rd.drop_duplicates()
print(calculate_processing_metrics(df_cleaned))
```

b. Polars

- Remove duplicated rows from the table data using unique().

```
[ ] df_cleaned = rd.unique()
print(calculate_processing_metrics(df_cleaned))
```

c. Modin

-Remove duplicated rows of data from the table using drop\_duplicates().

```
[ ] df_cleaned = rd.drop_duplicates()
print(calculate_processing_metrics(df_cleaned))
```

### 4. Remove missing data

a. Pandas

- Drop rows with missing values in the columns using dropna()

```
[ ] df_cleaned = df_cleaned.dropna()
    print(calculate_processing_metrics(df_cleaned))
```

b. Polars

- Drop rows with missing values in the columns using `drop_nulls()`.

```
[ ] df_cleaned = df_cleaned.drop_nulls()
    print(calculate_processing_metrics(df_cleaned))
```

c. Modin

- Drop rows with missing values in the columns using `dropna()`.

```
[ ] df_cleaned = df_cleaned.dropna()
    print(calculate_processing_metrics(df_cleaned))
```

## 5. Clean the teaser column

a. Pandas

- Clean the teaser column by removing unwanted characters by using `.str.replace(r'^a-zA-Z0-9: ,]', '')` and `.cleaned(pl.col('Teaser').str.contains(':'))`

```
[ ] df_cleaned['Teaser'] = df_cleaned['Teaser'].str.replace(r'^a-zA-Z0-9: ,]', '', regex=True)
    print(calculate_processing_metrics(df_cleaned))
```

```
[ ] df_cleaned = df_cleaned[df_cleaned['Teaser'].str.contains(':')]
    print(calculate_processing_metrics(df_cleaned))
```

b. Polars

- Clean the teaser column by removing unwanted characters by using `.str.replace_all(r'^a-zA-Z0-9: ,]', '')` and `.filter(pl.col('Teaser').str.contains(':'))`.

```
[ ] df_cleaned = df_cleaned.with_columns(
    pl.col('Teaser').str.replace_all(r'^a-zA-Z0-9: ,]', '')
)
    print(calculate_processing_metrics(df_cleaned))
```

```
[ ] df_cleaned = df_cleaned.filter(pl.col('Teaser').str.contains(':'))
    print(calculate_processing_metrics(df_cleaned))
```

### c. Modin

- Clean the teaser column by removing unwanted characters by using `.astype(str).str.replace(r'[^\a-zA-Z0-9: ,]', '', regex=True)` and `.str.contains(':')`.

```
[ ] df_cleaned['Teaser'] = df_cleaned['Teaser'].astype(str).str.replace(r'[^\a-zA-Z0-9: ,]', '', regex=True)
    print(calculate_processing_metrics(df_cleaned))
```

```
[ ] df_cleaned = df_cleaned[df_cleaned['Teaser'].str.contains(':')]
    print(calculate_processing_metrics(df_cleaned))
```

## 6. Splitting the place from the teaser column

### a. Pandas

- Get the place from the teaser column by using `.str.split(':', n=1, expand=True)`

```
[ ] df_cleaned[['Place', 'Teaser']] = df_cleaned['Teaser'].str.split(':', n=1, expand=True)
    print(calculate_processing_metrics(df_cleaned))

#KUALA LUMPUR: The Central Database Hub (PADU) system has recorded
# a total of 2.38 million individual information updates
```

### b. Polars

- Get the place from the teaser column by using `.str.split(':').list.get(0).alias('Place')` and `.str.split(':').list.get(1).alias('Teaser_New')`

```
[ ] # Split Teaser into Place and Teaser - corrected version
    df_cleaned = df_cleaned.with_columns([
        pl.col('Teaser').str.split(':').list.get(0).alias('Place'),
        pl.col('Teaser').str.split(':').list.get(1).alias('Teaser_New')
    ])

    # Replace the old Teaser with the new one
    df_cleaned = df_cleaned.with_columns([
        pl.col('Teaser_New').alias('Teaser')
    ]).drop('Teaser_New')

    print(calculate_processing_metrics(df_cleaned))

#KUALA LUMPUR: The Central Database Hub (PADU) system has recorded a total of 2.38 million individual information updates
```



### c. Modin

- Get the place from the teaser column by using `.str.split(':', n=1, expand=True)`

```
[ ] # Split "Teaser" into "Place" and "Teaser_New"
df_cleaned[['Place', 'Teaser_New']] = df_cleaned['Teaser'].str.split(':', n=1, expand=True)

print(calculate_processing_metrics(df_cleaned))

#KUALA LUMPUR: The Central Database Hub (PADU) system has recorded a total of 2.38 million individual information updates
```

## 7. Standardize place names

### a. Pandas

- Standardize the place names, convert them to uppercase, and remove any country names or other non-relevant information by using `.str.upper()`, `.str.split(',').str[0]` and `.str.replace(r'[^\a-zA-Z\s]+\s', '', regex=True)`. And extract it as a dictionary *known\_cities.csv*.

```
[ ] place_corrections = {
    'ALOR STAR': 'ALOR SETAR', 'AOR SETAR': 'ALOR SETAR', 'LOR STAR': 'ALOR SETAR', 'ASTANA KAZAKHSTAN': 'ASTANA',
    'BALIK PULAI': 'BALIK PULAU', 'BATANG AI': 'BATANG KALI', 'BAGAN DATOH': 'BAGAN DATUK',
    'CAMERON HIGHLAND': 'CAMERON HIGHLANDS', 'CHIANGMAI': 'CHIANG MAI', 'COLOMBO SRI LANKA': 'COLOMBO',
    'FRANK': 'FRANKFURT',
    'GUAMUSANG': 'GUA MUSANG', 'GUA MUSANG POS SIMPOR': 'GUA MUSANG',
    'DANANG': 'DA NANG',
    'GEORGE TOWN': 'GEORGE TOWN', 'GEORGETOWN': 'GEORGE TOWN', 'JERTIH': 'JERTEH',
    'JOHOR BARU': 'JOHOR BAHRU', 'JOHOR BAHU': 'JOHOR BAHRU', 'JOHOR BHARU': 'JOHOR BAHRU', 'JOHOR BARY': 'JOHOR BAHRU', 'JOHOR BAHRU': 'JOHOR BAHRU',
    'JOHOR BARU KUALA LUMPUR': 'JOHOR BAHRU', 'JOHOR BARUSINGAPORE': 'JOHOR BAHRU',
    'KUALA KUBU BARU': 'KUALA KUBU BAHRU', 'KUALA KUBU BAHRU': 'KUALA KUBU BAHRU',
    'UALA LUMPUR': 'KUALA LUMPUR', 'KUALKUALA LUMPUR': 'KUALA LUMPUR', 'SEPT KUALA LUMPUR': 'KUALA LUMPUR', 'KKUALA LUMUR': 'KUALA LUMPUR',
    'KIALA LUMPUR': 'KUALA LUMPUR', 'IKUALA LUMPUR': 'KUALA LUMPUR', 'KUALAA LUMPUR': 'KUALA LUMPUR', 'KUALALUMPUR': 'KUALA LUMPUR',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPU': 'KUALA LUMPUR', 'KUALA LUMPURHONG KONG': 'KUALA LUMPUR', 'KUALA LIMPUR': 'KUALA LUMPUR',
    'KUALA LUMPURJAKARTA': 'KUALA LUMPUR', 'KUALA KUMPUR': 'KUALA LUMPUR', 'KUALA NERUS TERENGGANU': 'KUALA NERUS',
    'KUALATERENGGANU': 'KUALA TERENGGANU', 'KUALA TERENGANU': 'KUALA TERENGGANU', 'KUALA TERENGAGNU': 'KUALA TERENGGANU', 'KUALA TENGGANU': 'KUALA TERENGGANU',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUCHINGL': 'KUCHING', 'KUANG': 'KLUANG',
    'KUAL LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPUR': 'KUALA LUMPUR',
    'UALA TERENGGANU': 'KUALA TERENGGANU', 'KKOTA KINABALU': 'KOTA KINABALU', 'KOTA KINABAU': 'KOTA KINABALU', 'KOTA KINABALU': 'KOTA KINABALU', 'KOTA KINBALU': 'KOTA KINABALU', 'KOTA KINABALU': 'KOTA KINABALU',
    'KOTA BARU': 'KOTA BAHRU', 'KOTA BAHRU': 'KOTA BAHRU', 'KOTA BARU': 'KOTA BAHRU', 'KOTA BARUGEOGE TOWN': 'KOTA BAHRU',
    'LABUAN BAJO INDONESIA': 'LABUAN BAJO', 'LONDONKUALA LUMPUR': 'LONDON', 'LONDON TUES': 'LONDON', 'LENGONG': 'LENGGONG', 'LANGKAWI': 'LANGKAWI',
    'MARNG': 'MARANG', 'MELAKA': 'MALACCA', 'MEKALA': 'MALACCA', 'MANAMA BAHRAIN': 'MANAMA',
    'NIBONG TEBAL': 'NIBONG TEBAL', 'NEW DELHI INDIA': 'NEW DELHI', 'NEW DELH': 'NEW DELHI', 'NARATHIAT SOUTHERN THAILAND': 'NARATHIAT', 'MUNDOK SOUTHERN THAILAND': 'MUNDOK',
    'PARISBEIJING': 'PARIS',
    'PUTRAJAYAS': 'PUTRAJAYA', 'PUTRAYAJA': 'PUTRAJAYA', 'PUTRAJAYA': 'PUTRAJAYA', 'PPUTRAJAYA': 'PUTRAJAYA', 'PATTANI THAILAND': 'PATTANI', 'PASIR PUTIH': 'PASIR PUTEH',
    'PORT MORESBY PAPUA NEW GUINEA': 'PORT MORESBY', 'PANGKOR ISLAND': 'PANGKOR', 'PULAU PERHENTIAN KECIL TERENGGANU': 'PULAU PERHENTIAN',
    'SEBERANG PERAI': 'SEBERANG PRAI', 'SUNNYLANDS CALIFORNIA': 'SUNNYLANDS', 'SUNGAI GOKOL THAILAND': 'SUNGAI GOKOL',
    'SUBANG': 'SUBANG JAYA', 'SONGKLA': 'SONGKHLA', 'SHAH ALAM': 'SHAH ALAM', 'SEMEYEH': 'SEMEYEH', 'SELANGAU': 'SELANGOR', 'SARI': 'SARIKEI',
    'SAMARAHAN': 'SAMARKAND', 'SADAO THAILAND': 'SADAO', 'ALSHAH ALAM': 'SHAH ALAM',
    'THE HAGUE NETHERLANDS': 'THE HAGUE', 'TASHKENTL': 'TASHKENT', 'TAKBAI SOUTHERN THAILAND': 'TAKBAI', 'TAK': 'TAK THAILAND',
    'VALLETTA MALTA': 'VALLETTA', 'VIENTIANE LAOS': 'VIENTIANE', 'VLADIVOSTOK RUSSIA': 'VLADIVOSTOK', 'VALETTA': 'VALLETTA',
    'ULAANBAATAR MONGOLIA': 'ULAANBAATAR', 'ULAANBAATAR MONGOLIA': 'ULAANBAATAR', 'ULAANBAATAR': 'ULAANBAATAR',
    'WASHINGTON DC': 'WASHINGTON', 'KKUALA LUMPUR': 'KUALA LUMPUR',
```

```
[ ] df_cleaned['Place'] = df_cleaned['Place'].str.upper()
df_cleaned['Place'] = df_cleaned['Place'].str.split(',').str[0]
df_cleaned['Place'] = df_cleaned['Place'].str.replace(r'[^\a-zA-Z\s]+\s', '', regex=True)
```

```
[ ] # Get unique place names and sort them
known_cities = pd.Series(df_cleaned['Place'].unique()).sort_values()
known_cities.to_csv('known_cities.csv', index=False, header=False)
```

```
[ ] df_cleaned = df_cleaned[df_cleaned['Place'].str.upper().isin(known_cities)]
print(calculate_processing_metrics(df_cleaned))
```

## b. Polars

- Standardize the place names, convert them to uppercase, and remove any country names or other non-relevant information by using `.str.to_uppercase()`, `.str.replace_all(old, new)`, `str.split(',').list.first()` and `.str.replace_all(r'[a-zA-Z\s]'+', '')`.

```
[ ] place_corrections = {
    'ALOR STAR': 'ALOR SETAR', 'AOR SETAR': 'ALOR SETAR', 'LOR STAR': 'ALOR SETAR', 'ASTANA KAZAKHSTAN': 'ASTANA',
    'BALIK PULAI': 'BALIK PULAU', 'BATANG AI': 'BATANG KALI', 'BAGAN DATOH': 'BAGAN DATUK',
    'CAMERON HIGHLAND': 'CAMERON HIGHLANDS', 'CHIANGMAI': 'CHIANG MAI', 'COLOMBO SRI LANKA': 'COLOMBO',
    'FRANK': 'FRANKFURT',
    'GUAMUSANG': 'GUA MUSANG', 'GUA MUSANG POS SIMPOR': 'GUA MUSANG',
    'DANANG': 'DA NANG',
    'GEORGE TOWN': 'GEORGE TOWN', 'GEORGETOWN': 'GEORGE TOWN', 'JERTIH': 'JERTEH',
    'JOHOR BARU': 'JOHOR BAHRU', 'JOHOR BAHU': 'JOHOR BAHRU', 'JOHOR BHARU': 'JOHOR BAHRU', 'JOHOR BARY': 'JOHOR BAHRU', 'JOHOR BAHRU': 'JOHOR BAHRU',
    'JOHOR BARU KUALA LUMPUR': 'JOHOR BAHRU', 'JOHOR BARUSINGAPORE': 'JOHOR BAHRU',
    'KUALA KUBU BARU': 'KUALA KUBU BAHRU', 'KUALA KUBU BAHRU': 'KUALA KUBU BAHRU',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUALKUALA LUMPUR': 'KUALA LUMPUR', 'SEPT KUALA LUMPUR': 'KUALA LUMPUR', 'KKUALA LUMPUR': 'KUALA LUMPUR',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'IKUALA LUMPUR': 'KUALA LUMPUR', 'KUALAA LUMPUR': 'KUALA LUMPUR', 'KUALALUMPUR': 'KUALA LUMPUR',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPU': 'KUALA LUMPUR', 'KUALA LUMPURHONG KONG': 'KUALA LUMPUR', 'KUALA LIMPUR': 'KUALA LUMPUR',
    'KUALA LUMPURJAKARTA': 'KUALA LUMPUR', 'KUALA KUMPUR': 'KUALA LUMPUR', 'KUALA NERUS TERENGGANU': 'KUALA NERUS',
    'KUALATERENGGANU': 'KUALA TERENGGANU', 'KUALA TERENGANU': 'KUALA TERENGGANU', 'KUALA TERENGAGNU': 'KUALA TERENGGANU', 'KUALA TENGGANU': 'KUALA TERENGGANU',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUCHINGL': 'KUCHING', 'KUANG': 'KLUANG',
    'KUAL LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPUR': 'KUALA LUMPUR',
    'UALA TERENGGANU': 'KUALA TERENGGANU', 'KKOTA KINABALU': 'KOTA KINABALU', 'KOTA KINABAU': 'KOTA KINABALU', 'KOTA KINBALU': 'KOTA KINABALU', 'KOTA KINABALU': 'KOTA KINABALU',
    'KOTA BARU': 'KOTA BAHRU', 'KOTA BAHRU': 'KOTA BAHRU', 'KOTA BARU': 'KOTA BAHRU', 'KOTA BARUGEOGE TOWN': 'KOTA BAHRU',
    'LABUAN BAJO INDONESIA': 'LABUAN BAJO', 'LONDONKUALA LUMPUR': 'LONDON', 'LONDON TUES': 'LONDON', 'LENGONG': 'LENGGONG', 'LANGKAWI': 'LANGKAWI',
    'MARNG': 'MARANG', 'MELAKA': 'MALACCA', 'MEKALA': 'MALACCA', 'MANAMA BAHRAIN': 'MANAMA',
    'NIBONG TEBU': 'NIBONG TEBAL', 'NEW DELHI INDIA': 'NEW DELHI', 'NEW DELH': 'NEW DELHI', 'NARATHIAT SOUTHERN THAILAND': 'NARATHIAT', 'MUNDOK SOUTHERN THAILAND': 'MUNDOK',
    'PARISBEIJING': 'PARIS',
    'PUTRAJAYAS': 'PUTRAJAYA', 'PUTRAYAJA': 'PUTRAJAYA', 'PUTRAJAYA': 'PUTRAJAYA', 'PPUTRAJAYA': 'PUTRAJAYA', 'PATTANI THAILAND': 'PATTANI', 'PASIR PUTIH': 'PASIR PUTEH',
    'PORT MORESBY PAPUA NEW GUINEA': 'PORT MORESBY', 'PANGKOR ISLAND': 'PANGKOR', 'PULAU PERHENTIAN KECIL TERENGGANU': 'PULAU PERHENTIAN',
    'SEBERANG PERAI': 'SEBERANG PRAI', 'SUNNYLANDS CALIFORNIA': 'SUNNYLANDS', 'SUNGAI GLOK THAILAND': 'SUNGAI GLOK',
    'SUBANG': 'SUBANG JAYA', 'SONGKLA': 'SONGKHLA', 'SHAH ALAM': 'SHAH ALAM', 'SEMEYEH': 'SEMEYEH', 'SELANGAU': 'SELANGOR', 'SARI': 'SARIKEI',
    'SAMARAHAN': 'SAMARAND', 'SADAO THAILAND': 'SADAO', 'ALSHAH ALAM': 'SHAH ALAM',
    'THE HAGUE NETHERLANDS': 'THE HAGUE', 'TASHKENTL': 'TASHKENT', 'TAKBAI SOUTHERN THAILAND': 'TAKBAI', 'TAK': 'TAK THAILAND',
    'VALLETTA MALTA': 'VALLETTA', 'VIENTIANE LAOS': 'VIENTIANE', 'VLADIVOSTOK RUSSIA': 'VLADIVOSTOK', 'VALETTA': 'VALLETTA',
    'ULAANBAATAR MONGOLIA': 'ULAANBAATAR', 'ULAANBAATAR MONGOLIA': 'ULAANBAATAR', 'ULAANBAATAR': 'ULAANBAATAR',
    'WASHINGTON DC': 'WASHINGTON', 'KKUALA LUMPUR': 'KUALA LUMPUR',
}
```

```
}

df_cleaned = df_cleaned.with_columns(
    pl.col('Place').str.to_uppercase()
)
for old, new in place_corrections.items():
    df_cleaned = df_cleaned.with_columns(
        pl.col('Place').str.replace_all(old, new)
    )
df_cleaned = df_cleaned.with_columns(
    pl.col('Place').str.split(',').list.first()
)
df_cleaned = df_cleaned.with_columns(
    pl.col('Place').str.replace_all(r'[a-zA-Z\s]'+', ''')
)

print(calculate_processing_metrics(df_cleaned))
```

```
[ ] # Count the number of articles per city
city_counts = (
    df_cleaned
        .group_by("Place")
        .agg(pl.len().alias("count"))
        .filter(pl.col("count") >= 2)
)

# Extract valid cities
valid_cities = city_counts["Place"]

# Save the valid cities to a CSV file
pl.DataFrame({"Place": valid_cities}).write_csv("valid_cities.csv")

# Filter the original DataFrame
df_cleaned = df_cleaned.filter(pl.col("Place").is_in(valid_cities))

print(calculate_processing_metrics(df_cleaned))
```

### c. Modin

- Standardize the place names, convert them to uppercase, and remove any country names or other non-relevant information by using `.str.upper()` , `.str.replace(old, new, regex=False)` , `.str.split(',').str[0]` and `.str.replace(r'^a-zA-Z\s|'+', ', regex=True)`.

```
[ ] place_corrections = {
    'ALOR STAR': 'ALOR SETAR', 'AOR SETAR': 'ALOR SETAR', 'LOR STAR': 'ALOR SETAR', 'ASTANA KAZAKHSTAN': 'ASTANA',
    'BALIK PULAI': 'BALIK PULAU', 'BATANG AI': 'BATANG KALI', 'BAGAN DATOH': 'BAGAN DATUK',
    'CAMERON HIGHLAND': 'CAMERON HIGHLANDS', 'CHIANGMAI': 'CHIANG MAI', 'COLOMBO SRI LANKA': 'COLOMBO',
    'FRANK': 'FRANKFURT',
    'GUAMUSANG': 'GUA MUSANG', 'GUA MUSANG POS SIMPOR': 'GUA MUSANG',
    'DANANG': 'DA NANG',
    'GEORGE TOWN': 'GEORGE TOWN', 'GEORGETOWN': 'GEORGE TOWN', 'JERTIH': 'JERTEH',
    'JOHOR BARU': 'JOHOR BAHRU', 'JOHOR BAHU': 'JOHOR BAHRU', 'JOHOR BHARU': 'JOHOR BAHRU', 'JOHOR BAHRU': 'JOHOR BAHRU', 'JOHOR BARY': 'JOHOR BAHRU', 'JOHOR BAHRU': 'JOHOR BAHRU',
    'JOHOR BARU KUALA LUMPUR': 'JOHOR BAHRU', 'JOHOR BARUSINGAPORE': 'JOHOR BAHRU',
    'KUALA KUBU BARU': 'KUALA KUBU BAHRU', 'KUALA KUBU BAHRU': 'KUALA KUBU BAHRU',
    'UALA LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPUR': 'KUALA LUMPUR', 'SEPT KUALA LUMPUR': 'KUALA LUMPUR', 'KKUALA LUMUR': 'KUALA LUMPUR',
    'KIALA LUMPUR': 'KUALA LUMPUR', 'IKUALA LUMPUR': 'KUALA LUMPUR', 'KUALAA LUMPUR': 'KUALA LUMPUR', 'KUALALUMPUR': 'KUALA LUMPUR',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPU': 'KUALA LUMPUR', 'KUALA LUMPUHONG KONG': 'KUALA LUMPUR', 'KUALA LIMPUR': 'KUALA LUMPUR',
    'KUALA LUMPURJAKARTA': 'KUALA LUMPUR', 'KUALA KUMPUR': 'KUALA LUMPUR', 'KUALA NERUS TERENGGANU': 'KUALA NERUS',
    'KUALATERENGGANU': 'KUALA TERENGGANU', 'KUALA TERENGANU': 'KUALA TERENGGANU', 'KUALA TERENGAGNU': 'KUALA TERENGGANU', 'KUALA TENGGANU': 'KUALA TERENGGANU',
    'KUALA LUMPUR': 'KUALA LUMPUR', 'KUCHINGL': 'KUCHING', 'KUANG': 'KLUANG',
    'KUAL LUMPUR': 'KUALA LUMPUR', 'KUALA LUMPUR': 'KUALA LUMPUR',
    'UALA TERENGGANU': 'KUALA TERENGGANU', 'KKOTA KINABALU': 'KOTA KINABALU', 'KOTA KINABAU': 'KOTA KINABALU', 'KOTA KINBALU': 'KOTA KINABALU', 'KOTA KINABALU': 'KOTA KINABALU',
    'KOTA BARU': 'KOTA BAHRU', 'KOTA BAHARU': 'KOTA BAHRU', 'KOTA BARU': 'KOTA BAHRU', 'KOTA BARUGEOGE TOWN': 'KOTA BAHRU',
    'LABUAN BAJO INDONESIA': 'LABUAN BAJO', 'LONDONKUALA LUMPUR': 'LONDON', 'LONDON TUES': 'LONDON', 'LENGONG': 'LENGGONG', 'LANGKAWI': 'LANGKAWI',
    'MARNG': 'MARANG', 'MELAKA': 'MALACCA', 'MEKALA': 'MALACCA', 'MANAMA BAHRAIN': 'MANAMA',
    'NIBONG TEBAL': 'NIBONG TEBAL', 'NEW DELHI INDIA': 'NEW DELHI', 'NEW DELH': 'NEW DELHI', 'NARATHIWAT SOUTHERN THAILAND': 'NARATHIWAT', 'MUNDOK SOUTHERN THAILAND': 'MUNDOK',
    'PARISBEIJING': 'PARIS',
    'PUTRAJAYAS': 'PUTRAJAYA', 'PUTRAYAJA': 'PUTRAJAYA', 'PUTRJAYA': 'PUTRAJAYA', 'PPUTRAJAYA': 'PUTRAJAYA', 'PATTANI THAILAND': 'PATTANI', 'PASIR PUTIH': 'PASIR PUTEH',
    'PORT MORESBY PAPUA NEW GUINEA': 'PORT MORESBY', 'PANGKOR ISLAND': 'PANGKOR', 'PULAU PERHENTIAN KECIL TERENGGANU': 'PULAU PERHENTIAN',
    'SEBERANG PERAI': 'SEBERANG PRAI', 'SUNNYLANDS CALIFORNIA': 'SUNNYLANDS', 'SUNGAI GLOK THAILAND': 'SUNGAI GLOK',
    'SUBANG': 'SUBANG JAYA', 'SONGKLA': 'SONGKHLA', 'SHAH ALAM': 'SHAH ALAM', 'SEMEHYEH': 'SEMEHYEH', 'SELANGAU': 'SELANGOR', 'SARI': 'SARIKEI',
    'SAMARAHAN': 'SAMARKAND', 'SADAO THAILAND': 'SADAO', 'ALSHAH ALAM': 'SHAH ALAM',
    'THE HAGUE NETHERLANDS': 'THE HAGUE', 'TASHKENTL': 'TASHKENT', 'TAKBAI SOUTHERN THAILAND': 'TAKBAI', 'TAK': 'TAK THAILAND',
    'VALLETTA MALTA': 'VALLETTA', 'VIENTIANE LAOS': 'VIENTIANE', 'VLADIVOSTOK RUSSIA': 'VLADIVOSTOK', 'VALETTA': 'VALLETTA',
    'ULANBAATAR MONGOLIA': 'ULANBAATAR', 'ULANBAATAR MONGOLIA': 'ULANBAATAR', 'ULANBAATAR': 'ULANBAATAR',
    'WASHINGTON DC': 'WASHINGTON', 'KUALA LUMPUR': 'KUALA LUMPUR',
}
```

```
}

df_cleaned['Place'] = df_cleaned['Place'].str.upper()
for old, new in place_corrections.items():
    df_cleaned['Place'] = df_cleaned['Place'].str.replace(old, new, regex=False)
df_cleaned['Place'] = df_cleaned['Place'].str.split(',').str[0]
df_cleaned['Place'] = df_cleaned['Place'].str.replace(r'^a-zA-Z\s|'+', ', regex=True)

print(calculate_processing_metrics(df_cleaned))
```

```
[ ] # Count the number of articles per city
city_counts = df_cleaned['Place'].value_counts()

# Set a threshold: keep only cities with at least N articles
threshold = 2
valid_cities = city_counts[city_counts >= threshold].index

# Save the valid cities to a CSV file
pd.Series(valid_cities).to_csv('valid_cities.csv', index=False, header=False)

# Filter the DataFrame to keep only valid cities
df_cleaned = df_cleaned[df_cleaned['Place'].isin(valid_cities)]

# Print processing metrics
print(calculate_processing_metrics(df_cleaned))
```

## 8. Extract date from URL

### a. Pandas

- Extract the date in YYYY/MM format from the URL and add it as a separate column in the dataset by using `.str.extract(r'(\d{4}/\d{2})')`.

```
[ ] df_cleaned['Date'] = df_cleaned['URL'].str.extract(r'(\d{4}/\d{2})')
print("Date column extracted from the URL.")
print(calculate_processing_metrics(df_cleaned))
```

### b. Polars

- Extract the date in YYYY/MM format from the URL and add it as a separate column in the dataset by using `.str.extract(r"(\d{4}/\d{2})").alias("Date")`.

```
[ ] df_cleaned = df_cleaned.with_columns(
    pl.col("URL").str.extract(r"(\d{4}/\d{2})").alias("Date")
)

print("Date column extracted from the URL.")

print(calculate_processing_metrics(df_cleaned))
```

### c. Modin

- Extract the date in YYYY/MM format from the URL and add it as a separate column in the dataset by using `.str.extract(r'(\d{4}/\d{2})')`.

```
[ ] df_cleaned['Date'] = df_cleaned['URL'].str.extract(r'(\d{4}/\d{2})')
print("Date column extracted from the URL.")
print(calculate_processing_metrics(df_cleaned)) # Metrics after extracting the Date column
print("")
```

## 6.0 Performance Evaluation

### 6.1. Before vs after optimization

Data processing was first performed using Pandas, which operates in a single-threaded manner. Although Pandas offers a user-friendly API, its lack of parallelism leads to performance bottlenecks with large datasets. Tasks such as data loading, duplicate removal, missing value handling, string cleaning, column splitting, place name standardization and date extraction were used to establish baseline metrics. The same tasks were re-executed using Polars and Modin. Polars offers high performance through efficient memory use and native multi-core parallelism while Modin speeds up Pandas workflows by distributing tasks across CPU cores. Performance improvements were measured by comparing how each library processed the same large-scale dataset with the average calculated from three measurements.

### 6.2. Processing time, CPU usage, memory usage and throughput

#### 1. Loading Data

Performance Metrics	Pandas	Polars	Modin
Total Processing Time (seconds)	1.1660	1.0009	1.0010
Initial CPU Usage (%)	38.20	58.10	57.13
Final CPU Usage (%)	23.07	46.27	58.20
Memory Usage (%)	21.37	21.80	21.97
Throughput (records/sec)	94922.15	110547.61	110531.92

## 2. Handle Duplicate Data

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1372	1.0006	1.0733
Initial CPU Usage (%)	10.20	12.33	74.47
Final CPU Usage (%)	5.17	5.37	5.50
Memory Usage (%)	21.33	21.77	22.03
Throughput (records/sec)	93631.39	106407.09	99201.30

## 3. Handle Missing Data

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1954	1.0005	1.0723
Initial CPU Usage (%)	6.90	4.87	25.70
Final CPU Usage (%)	4.90	6.20	4.70
Memory Usage (%)	21.33	21.77	22.00
Throughput (records/sec)	88167.97	105335.91	98292.13

#### 4. Clean the Teaser Column

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1826	1.0006	1.0007
Initial CPU Usage (%)	5.07	4.70	47.97
Final CPU Usage (%)	4.87	5.20	4.40
Memory Usage (%)	21.33	21.77	22.03
Throughput (records/sec)	89118.82	105331.54	105321.33

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1944	1.0006	1.0445
Initial CPU Usage (%)	10.70	5.23	46.37
Final CPU Usage (%)	29.37	4.53	5.03
Memory Usage (%)	21.30	21.77	21.97
Throughput (records/sec)	86350.94	103003.67	98679.35

5. Splitting the place from 'Teaser' column

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.2424	1.0005	1.0007
Initial CPU Usage (%)	41.60	5.53	10.30
Final CPU Usage (%)	31.47	28.30	38.43
Memory Usage (%)	21.37	21.77	21.97
Throughput (records/sec)	83065.61	103011.94	102997.50

6. Extract and Standardize Place Names

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1889	1.0007	1.0010
Initial CPU Usage (%)	5.03	39.80	100.00
Final CPU Usage (%)	5.03	8.70	100.00
Memory Usage (%)	21.33	21.83	21.97
Throughput (records/sec)	86694.45	103000.17	102964.66

<b>Performance Metrics</b>	<b>Pandas</b>	<b>Polars</b>	<b>Modin</b>
Total Processing Time (seconds)	1.1886	1.0006	1.0609



Initial CPU Usage (%)	4.93	5.57	38.77
Final CPU Usage (%)	5.20	4.33	28.70
Memory Usage (%)	21.33	21.80	21.37
Throughput (records/sec)	86463.74	102698.85	96861.25

Performance Metrics	Pandas	Polars	Modin
Total Processing Time (seconds)	1.1917	1.0006	1.1003
Initial CPU Usage (%)	6.07	5.70	76.27
Final CPU Usage (%)	6.00	5.50	55.57
Memory Usage (%)	21.33	21.80	21.60
Throughput (records/sec)	86237.43	102698.47	93389.56

#### 7. Extract Date from URL

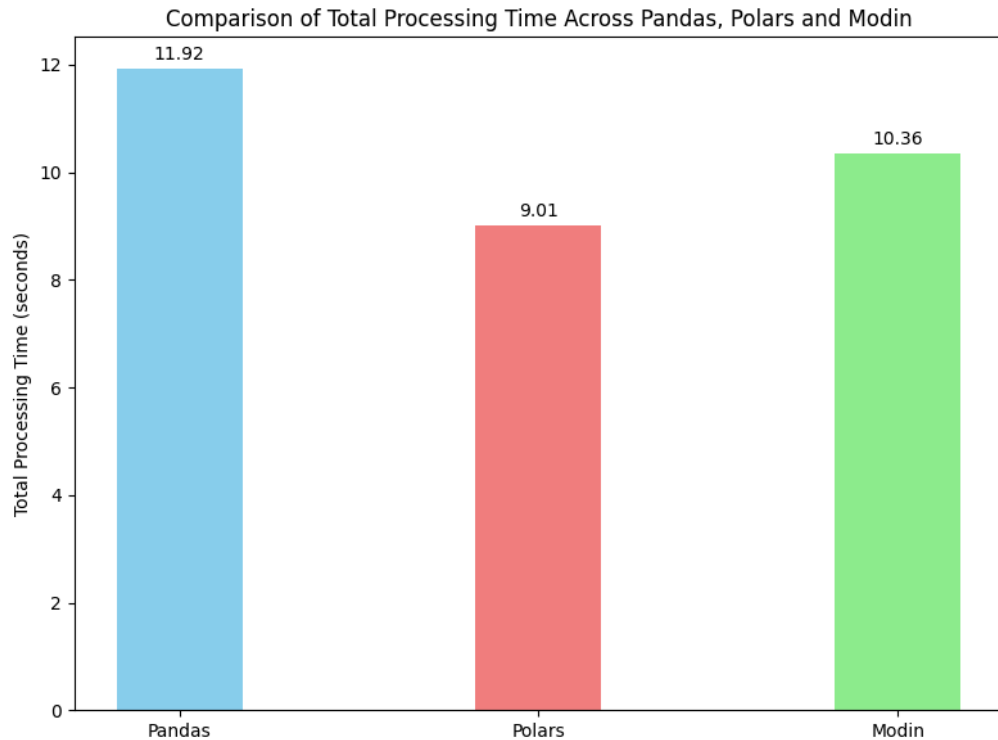
Performance Metrics	Pandas	Polars	Modin
Total Processing Time (seconds)	1.2322	1.0008	1.0007
Initial CPU Usage (%)	22.50	6.43	54.40
Final CPU Usage (%)	41.43	4.67	5.90
Memory Usage	21.40	21.80	21.30

(%)			
Throughput (records/sec)	83463.20	102673.35	102680.96

### 6.3. Charts and Graphs

To visually summarize the performance differences between Pandas, Polars and Modin, a series of bar charts were generated based on the aggregated performance metrics measured during the data processing.

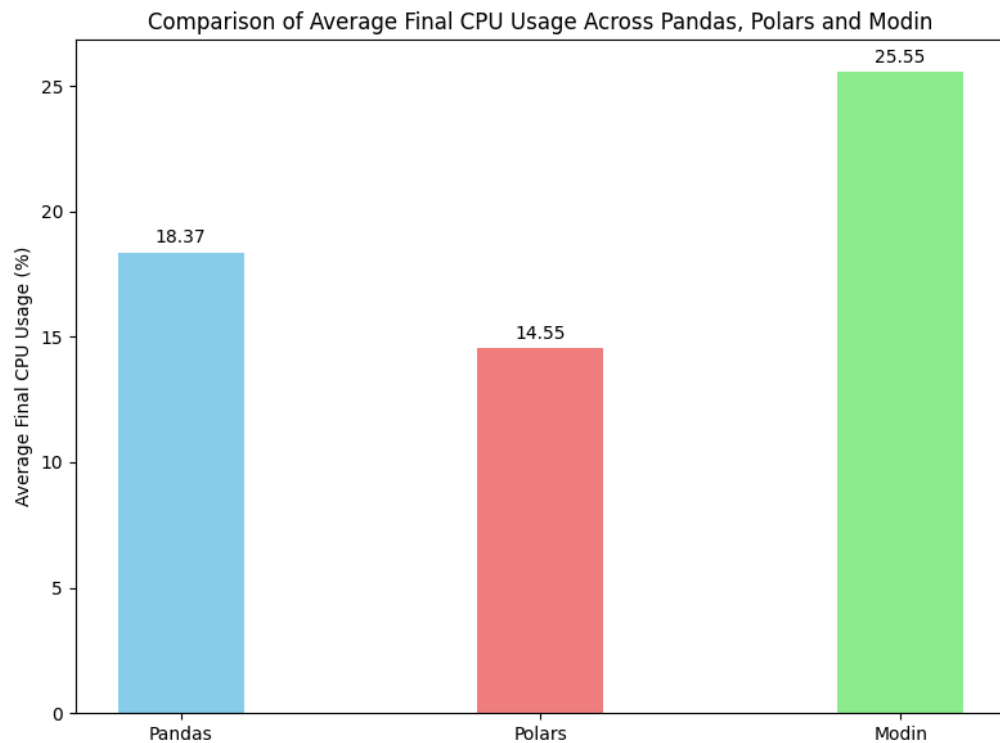
#### 1. Comparison of Total Processing Time



***Figure 6.3.1: Comparison of Total Processing Time Across Pandas, Polars and Modin***

The chart clearly indicates that Polars completed the entire set of data processing tasks in the shortest time followed by Modin and Pandas. This highlights the efficiency gains from the parallel processing capabilities inherent in Polars and utilized by Modin's backend, compared to the single-threaded execution model of Pandas.

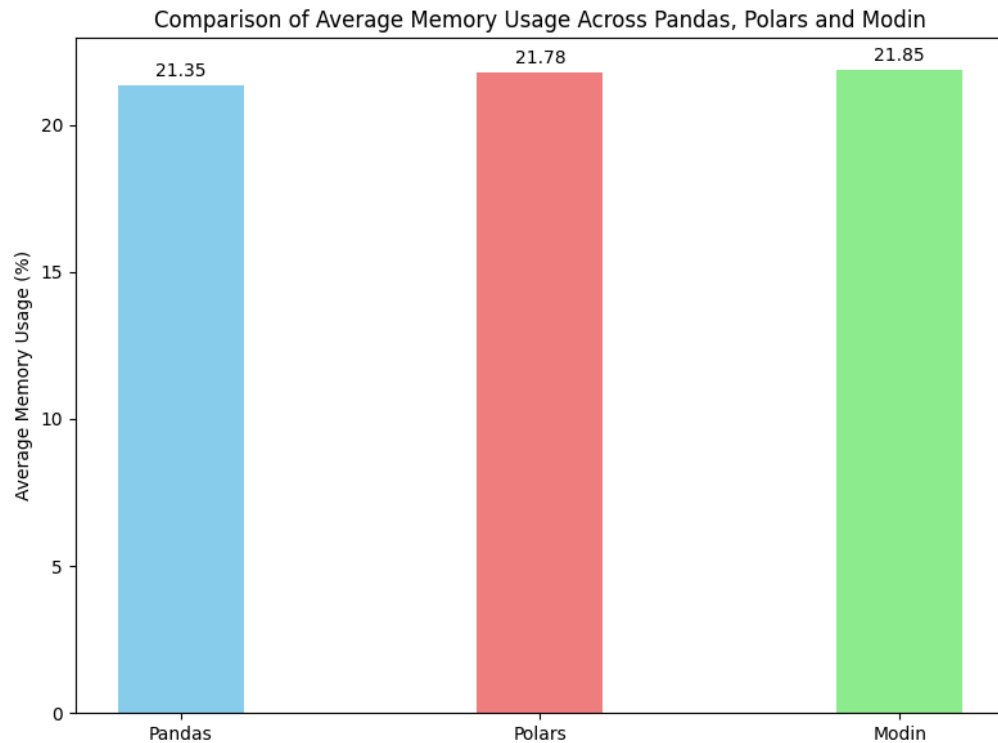
## 2. Comparison of Average Final CPU Usage



***Figure 6.3.2: Comparison of Average Final CPU Usage Across Pandas, Polars and Modin***

The chart shows Modin achieving the highest average final CPU usage. This suggests that Modin effectively utilized multiple CPU cores for its parallel tasks. Meanwhile, Pandas has the next highest average final CPU usage. Since it is single-threaded in nature, this suggests the active core was likely working intensively during processing, even though the overall system usage stayed low. Although Polars is the fastest library, it showed the lowest average final CPU usage among the three libraries. This indicates that Polars' operations are not only parallel but also highly optimized in which it achieves significant speedups without needing as many CPU resources as Modin.

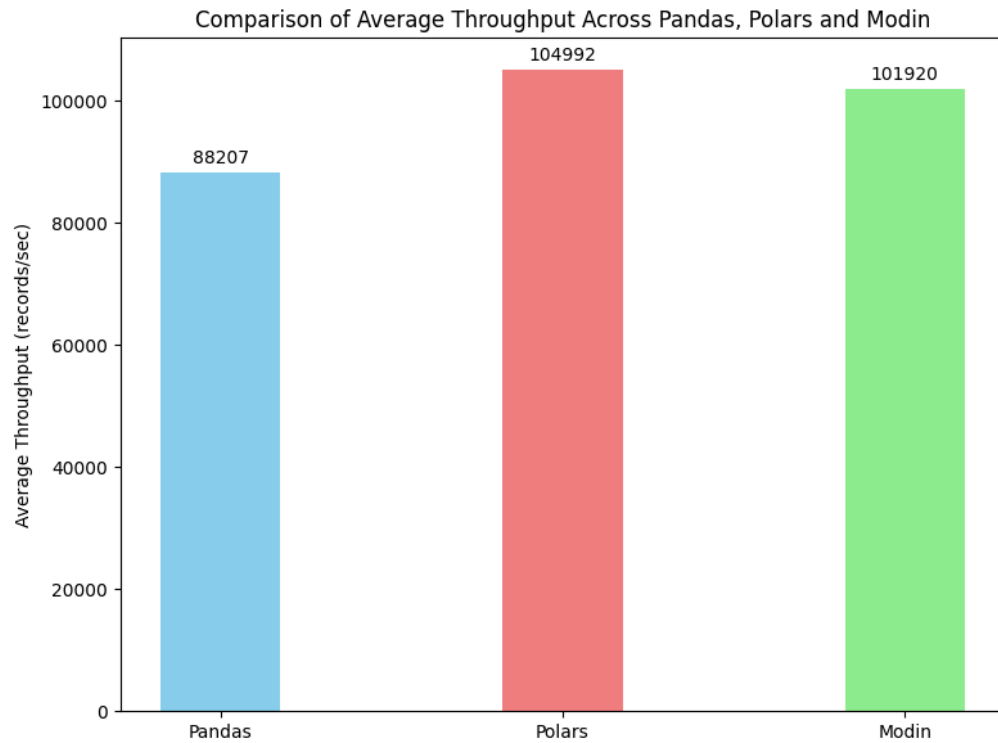
### 3. Comparison of Average Memory Usage



***Figure 6.3.3: Comparison of Average Memory Usage Across Pandas, Polars and Modin***

The chart shows Pandas has the lowest average memory usage. Polars was next, followed by Modin, which had the highest average memory usage. While Polars' efficiency is likely due to its Rust backend and Arrow-based memory layout, the aggregated average here is very close to Pandas. Even though Pandas does not use parallel processing, it has a mature and generally efficient memory model for data processing. The slightly higher figures for Polars and Modin in this averaged context could be due to the memory overhead of their respective backends, management of data partitions across cores and parallel execution. All three libraries used memory within a relatively tight range when processing the dataset.

#### 4. Comparison of Average Throughput



***Figure 6.3.4: Comparison of Average Throughput Across Pandas, Polars and Modin***

The chart shows Polars has the highest average throughput. Modin followed with a strong throughput while Pandas had the lowest. Throughput is directly related to processing speed. Thus, libraries that process data faster will naturally have a higher throughput. Both Polars and Modin significantly outperformed Pandas in terms of the number of records processed per second.

## 7.0 Challenges & Limitations

During this project, several challenges were encountered and certain limitations of the developed solution were identified:

1. **Initial Target Website Infeasibility:** Originally, the project aimed to scrape the Lazada website due to its extensive and valuable data. However, we encountered significant difficulties in successfully scraping Lazada. This challenge, along with the fact that another group had also chosen Lazada, led to a strategic decision to switch our target to the New Straits Times (NST) website.
2. **Tooling and Environment Constraints:** We initially considered using Scrapy, a powerful web scraping framework. However, we found that Scrapy was not fully compatible with Google Colab, our preferred development environment, because processing a large dataset (aiming for over 100,000 records) on local systems will have a potential resource constraint. Therefore, we aimed to overcome it by using a cloud-based environment. As a result, we opted to use BeautifulSoup (in conjunction with Playwright and Asyncio) for scraping.
3. **Anti-Scraping Measures:** Websites often employ anti-scraping measures to protect their content. While our implemented strategies such as rate-limiting and human-like interaction patterns aimed to navigate these on the NST website, such defenses are indeed a challenge in web scraping. They can limit the speed of data collection and require ongoing adjustments if the target website modifies its protective mechanisms.
4. **Data Quality and Inconsistency:** The NST website, particularly the older articles from 2014 to 2019, has data quality issues. These included human typographical errors, non-standardized data formats and inconsistencies in how information was presented such as the variations in place names like "Johor Baharu". This data noisiness required more intensive effort during the data cleaning phase and represents a limitation in the raw data's initial quality.

## **8.0 Conclusion & Future Work**

### **8.1. Summary of findings**

To conclude, in this project we had successfully designed and built a web scraper by using tools like Playwright, BeautifulSoup and Asyncio to collect more than 110,000 news article data from pages of the New Straits Times website. For web scraping, Playwright is used to automate browsers, navigate to web pages and click buttons to get the HTML pages. BeautifulSoup is used to parse the HTML page and extract the data like title, teaser, URL and category by searching the HTML page. Asyncio is used to improve the process of web scraping by handling many tasks concurrently.

After collecting the data, we clean and process the data to make it more organized, as the data collected from web scraping is raw and not organized. We had used 3 libraries and tools, which are Pandas, Polars and Modin in Python, to process the data. We then compared these 3 different libraries and tools in time, memory, CPU usage and throughput. We use Pandas as our traditional library for data processing and Polars and Modin libraries to provide optimization methods like multithreading and multiprocessing to improve the speed and performance of data processing. As for Pandas, it works slower with large datasets as it processes data in a single thread. For Polars, it is the fastest way as it uses multithreading to handle many tasks at a time. Lastly, Modin also improved performance by using multiprocessing to split tasks across multiple CPU cores. As a result, Polars and Modin are better choices for handling large data compared to pandas. This is because Polars and Modin provide optimization methods that can improve the performance of data processing. So, using the correct tools is important as the optimized tools help to improve the performance and reduce the time to process big data.



## **8.2. What could be improved**

In this project, we learned many skills in web scraping and data processing and faced many problems and challenges while handling the data. The challenges that we faced were website restriction, inconsistent data and technical issues with the tools that we used in Google Colab. There are still many things that can be improved in this project. The improvements are:

### **1. Save the collected and cleaned data into a database**

- Now we save the data into CSV files for simple storage. But it would be better to store the data in databases like PostgreSQL, MongoDB and NoSQL. This is because databases can handle large data efficiently and with better organization.

### **2. Use distributed computing for web scraping**

- This is to run multiple scrapers at the same time across many different machines. For example, we can use the Scrapy cluster to run many tasks in parallel.

### **3. Improve fault tolerance in web scraping**

- If the code suddenly crashes or stops running in the middle due to some external factor, all the progress might be lost. So, saving progress frequently is important for scraping for a long time.

### **4. Process data in chunks**

- Loading large CSV files at one time could be heavy. So, processing the data in chunks by setting the chunk size could improve the performance of data processing.

## References

IBM. (2024, July 9). *HPC*. Ibm.com. <https://www.ibm.com/think/topics/hpc>

NetApp. (n.d.). *What Is High-Performance Computing (HPC)? How It Works | NetApp*.

[Www.netapp.com](http://www.netapp.com).

<https://www.netapp.com/data-storage/high-performance-computing/what-is-hpc/>

Perez, M. (2019, August 6). *What is Web Scraping and What is it Used For? | ParseHub*.

ParseHub Blog. <https://www.parsehub.com/blog/what-is-web-scraping/>

Wikipedia Contributors. (2019, October 4). *Web scraping*. Wikipedia; Wikimedia Foundation.

[https://en.wikipedia.org/wiki/Web\\_scraping](https://en.wikipedia.org/wiki/Web_scraping)

## Appendices

### Sample Code Snippets

#### Web Scraper

```
MAX_PAGE_TIMEOUT = 600_000 # Increased timeout for slow loading pages
RETRY_WAIT = 10_000
MAX_RECORDS = 6_000
NUM_WORKERS = 8 # Number of concurrent tabs
PAGES_PER_WORKER = 1000 # How many pages each tab handles

lock = asyncio.Lock() # for writing to CSV from multiple workers

async def scrape_page(page, url):
    max_retries = 3
    for attempt in range(max_retries):
        try:
            print(f"[{url}] Attempt {attempt + 1}")
            await page.goto(url, timeout=MAX_PAGE_TIMEOUT, wait_until="domcontentloaded")

            # Increase wait time to allow more time for page content to load
            await page.wait_for_timeout(5000 + random.randint(4000, 8000)) # Increased wait time
            return await page.content()
        except Exception as e:
            print(f"⚠ Error loading {url}: {e}")
            if attempt == max_retries - 1:
                return None
            await page.wait_for_timeout(RETRY_WAIT)

async def worker(browser, writer, start_page, end_page, worker_id):
    total_scraped = 0
    page = await browser.new_page()

    for page_num in range(start_page, end_page + 1):
        if total_scraped >= MAX_RECORDS // NUM_WORKERS:
            break

        url = f'https://www.nst.com.my/news/nation?page={page_num}'
        html = await scrape_page(page, url)
        if not html:
            continue

        soup = BeautifulSoup(html, 'html.parser')
        articles = soup.find_all('a', class_='d-flex article listing mb-3 pb-3')

        if not articles:
            print(f"🚫 Worker {worker_id}: No articles on page {page_num}")
            continue

        records = []
        for article in articles:
            if total_scraped >= MAX_RECORDS // NUM_WORKERS:
                break

            title_tag = article.find('h6', class_='field-title')
            teaser_tag = article.find('div', class_='d-block article-teaser')
            link_tag = article.get('href')
            category_tag = article.find('span', class_='field-category')
```

*Figure 7: Web Scraper Source Code Part 1*

```

        title = title_tag.get_text(strip=True) if title_tag else 'No Title'
        teaser = teaser_tag.get_text(strip=True) if teaser_tag else 'No Teaser'
        full_url = f"https://www.nst.com.my{link_tag}" if link_tag else 'No URL'
        category = category_tag.get_text(strip=True) if category_tag else 'No Category'

        records.append([title, full_url, teaser, category])
        total_scraped += 1

    async with lock:
        writer.writerows(records)

    print(f"✅ Worker {worker_id}: Page {page_num} done, total: {total_scraped}")

    await page.close()

async def run():
    with open('nst_nation2.csv', 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['Title', 'URL', 'Teaser', 'Category'])

    async with async_playwright() as p:
        browser = await p.chromium.launch(headless=True)
        tasks = []

        for i in range(NUM_WORKERS):
            start_page = i * PAGES_PER_WORKER + 1
            end_page = (i + 1) * PAGES_PER_WORKER
            task = worker(browser, writer, start_page, end_page, worker_id=i+1)
            tasks.append(task)

        await asyncio.gather(*tasks)
        await browser.close()

    print(f"🎉 Scraping complete! {MAX_RECORDS} records across {NUM_WORKERS} workers.")

# Run the async function
await run()

```

**Figure 8: Web Scraper Source Code Part 2**

## Charts and Graphs for Performance Evaluation

```
import matplotlib.pyplot as plt
import numpy as np

libraries = ['Pandas', 'Polars', 'Modin']

# Overall Total Processing Time for the entire pipeline
total_times = [11.9194, 9.0064, 10.3554] # seconds

# Overall Average Final CPU Usage across all operations
avg_final_cpu_usage = [18.37, 14.55, 25.55] # %

# Overall Average Memory Usage across all operations
avg_memory_usage = [21.35, 21.78, 21.85] # %

# Overall Average Throughput across all operations
avg_throughput = [88207.20, 104991.81, 101920.38] # records/sec

# --- Plotting ---
x = np.arange(len(libraries)) # the label locations
bar_width = 0.35 # the width of the bars (can adjust if needed, e.g., 0.25 or 0.4)
colors = ['skyblue', 'lightcoral', 'lightgreen'] # Consistent colors

# Plot 1: Total Processing Time
fig1, ax1 = plt.subplots(figsize=(8, 6))
rects1 = ax1.bar(x, total_times, bar_width, label='Time', color=colors)
ax1.set_ylabel('Total Processing Time (seconds)')
ax1.set_title('Comparison of Total Processing Time Across Pandas, Polars and Modin')
ax1.set_xticks(x)
ax1.set_xticklabels(libraries)
ax1.bar_label(rects1, fmt='%.2f', padding=3)
fig1.tight_layout()
plt.savefig('comparison_total_processing_time.png')
plt.show()

# Plot 2: Average Final CPU Usage
fig2, ax2 = plt.subplots(figsize=(8, 6))
```

**Figure 9: Performance Evaluation Charts (1)**

```

rects2 = ax2.bar(x, avg_final_cpu_usage, bar_width, label='CPU Usage', color=colors)
ax2.set_ylabel('Average Final CPU Usage (%)')
ax2.set_title('Comparison of Average Final CPU Usage Across Pandas, Polars and Modin')
ax2.set_xticks(x)
ax2.set_xticklabels(libraries)
ax2.bar_label(rects2, fmt='%.2f', padding=3)
fig2.tight_layout()
plt.savefig('comparison_avg_final_cpu_usage.png')
plt.show()

# Plot 3: Average Memory Usage
fig3, ax3 = plt.subplots(figsize=(8, 6))
rects3 = ax3.bar(x, avg_memory_usage, bar_width, label='Memory Usage', color=colors)
ax3.set_ylabel('Average Memory Usage (%)')
ax3.set_title('Comparison of Average Memory Usage Across Pandas, Polars and Modin')
ax3.set_xticks(x)
ax3.set_xticklabels(libraries)
ax3.bar_label(rects3, fmt='%.2f', padding=3)
fig3.tight_layout()
plt.savefig('comparison_avg_memory_usage.png')
plt.show()

# Plot 4: Average Throughput
fig4, ax4 = plt.subplots(figsize=(8, 6))
rects4 = ax4.bar(x, avg_throughput, bar_width, label='Throughput', color=colors)
ax4.set_ylabel('Average Throughput (records/sec)')
ax4.set_title('Comparison of Average Throughput Across Pandas, Polars and Modin')
ax4.set_xticks(x)
ax4.set_xticklabels(libraries)
ax4.bar_label(rects4, fmt='%0f', padding=3) # Format as integer for throughput
fig4.tight_layout()
plt.savefig('comparison_avg_throughput.png')
plt.show()

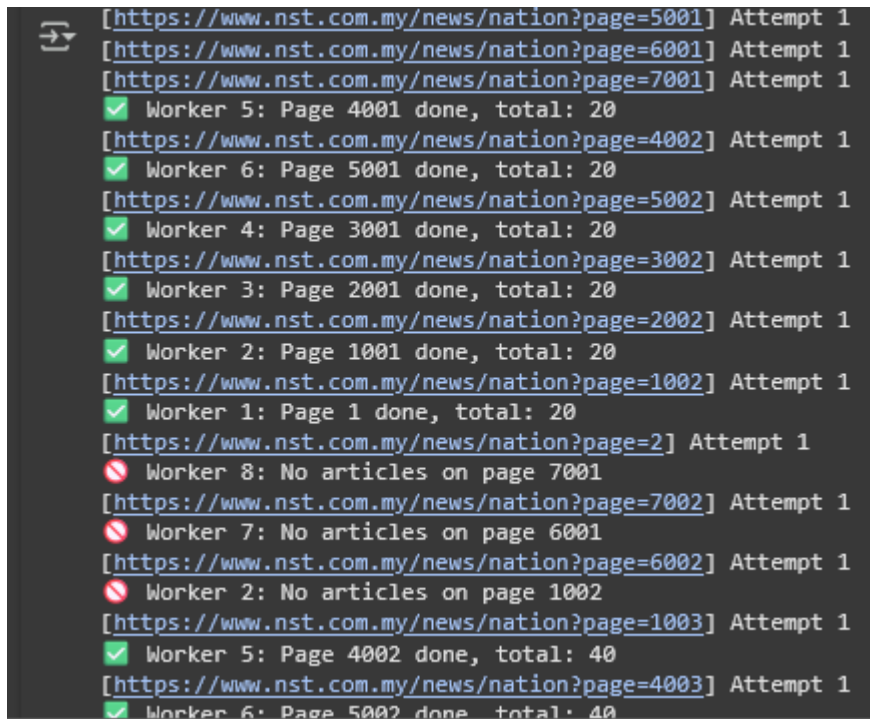
print("Charts generated and saved as PNG files.")

```

***Figure 10: Performance Evaluation Charts (2)***

## Screenshots of output

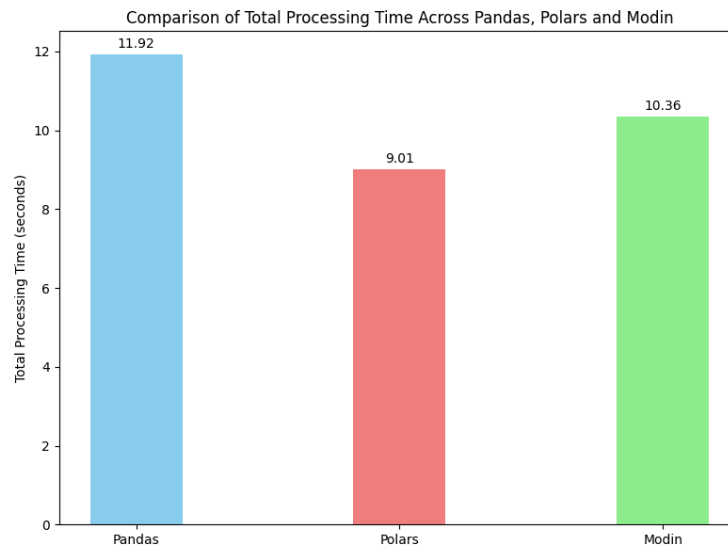
### Web Scraper



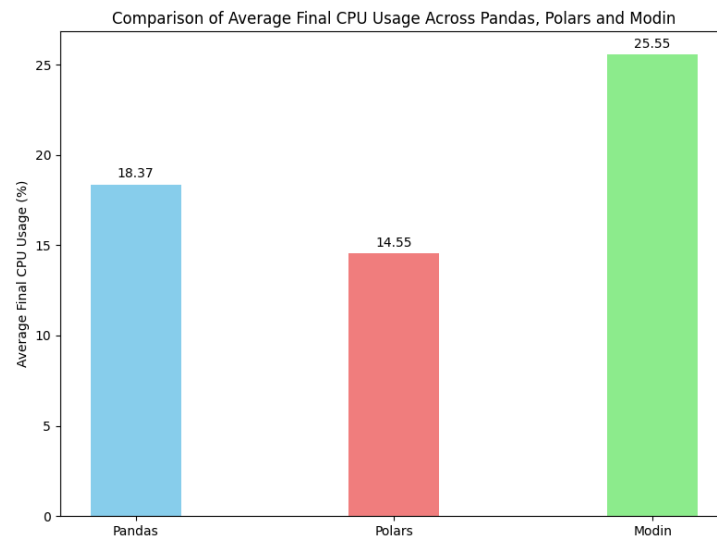
```
[https://www.nst.com.my/news/nation?page=5001] Attempt 1
[https://www.nst.com.my/news/nation?page=6001] Attempt 1
[https://www.nst.com.my/news/nation?page=7001] Attempt 1
✓ Worker 5: Page 4001 done, total: 20
[https://www.nst.com.my/news/nation?page=4002] Attempt 1
✓ Worker 6: Page 5001 done, total: 20
[https://www.nst.com.my/news/nation?page=5002] Attempt 1
✓ Worker 4: Page 3001 done, total: 20
[https://www.nst.com.my/news/nation?page=3002] Attempt 1
✓ Worker 3: Page 2001 done, total: 20
[https://www.nst.com.my/news/nation?page=2002] Attempt 1
✓ Worker 2: Page 1001 done, total: 20
[https://www.nst.com.my/news/nation?page=1002] Attempt 1
✓ Worker 1: Page 1 done, total: 20
[https://www.nst.com.my/news/nation?page=2] Attempt 1
✗ Worker 8: No articles on page 7001
[https://www.nst.com.my/news/nation?page=7002] Attempt 1
✗ Worker 7: No articles on page 6001
[https://www.nst.com.my/news/nation?page=6002] Attempt 1
✗ Worker 2: No articles on page 1002
[https://www.nst.com.my/news/nation?page=1003] Attempt 1
✓ Worker 5: Page 4002 done, total: 40
[https://www.nst.com.my/news/nation?page=4003] Attempt 1
✓ Worker 6: Page 5002 done, total: 40
```

*Figure 11: Web Scraper Output*

## Charts and Graphs for Performance Evaluation

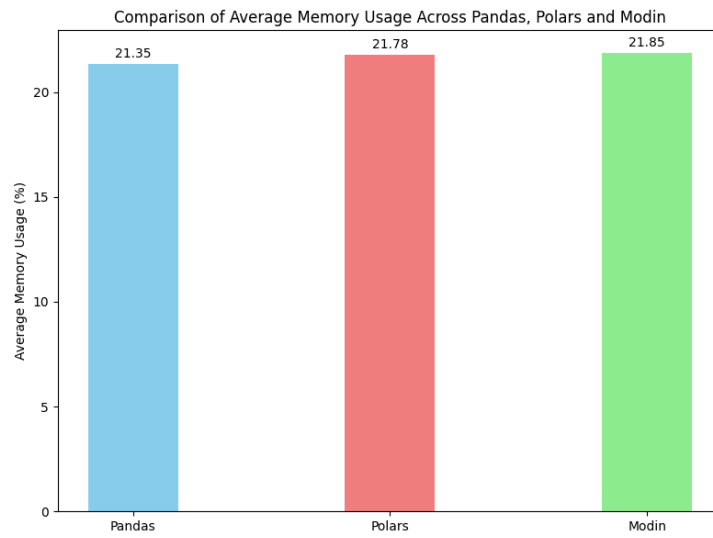


**Figure 12: Chart (1)**

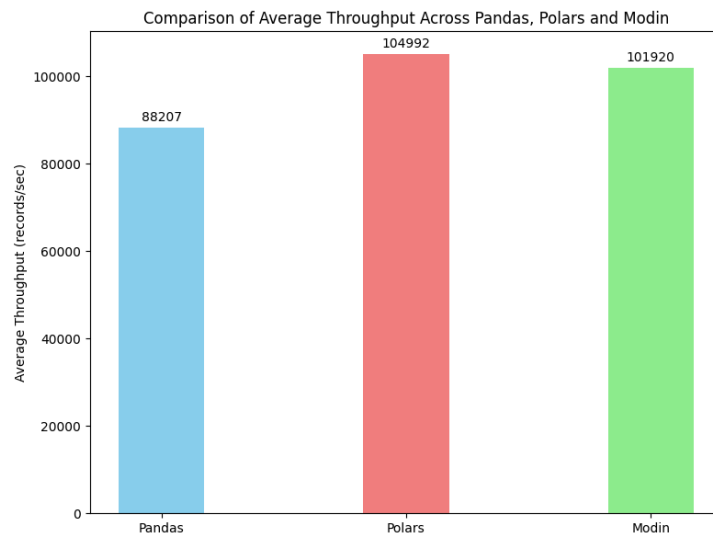


**Figure 13: Chart (2)**





**Figure 14: Chart (3)**



**Figure 15: Chart (4)**

**Links to full code repo or dataset**

Link : <https://github.com/drshahizan/HPDP/tree/main/2425/project/p1/GroupE>