Project 1

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

SECP3133

HIGH PERFORMANCE DATA PROCESSING

NAME	MATRIC NUMBER
CHEN PYNG HAW	A22EC0042
MUHAMMAD DANIAL BIN AHMAD SYAHIR	A22EC0206
LOW JIE SHENG	A22EC0075
NADHRAH NURSABRINA BINTI ZULAINI	A22EC0224

Submission date: 16/5/2025

Table of Contents

1.0 Introduction

1.1 Background of the Project

In this digital era, the big data explosion has changed the ways in which industries operate and make decisions. The rise of big data and web-based information extraction has revolutionized market analysis, customer sentiment tracking, business intelligence, and content aggregation. The efficient retrieval of large-scale web data opens doors for both organizations and researchers to observe the trends and garner actionable insights to be shared in the form of firm decisions. This project aims to develop a highly optimized system for extracting and processing large datasets from The Edge Malaysia, thus feeding into the demand for scalable automated data collection and processing pipelines.

This project focuses on studying the web scraping techniques to retrieve the large scale of data from The Edge Malaysia, which is a popular Malaysian news website. The main goal of the project is to create and optimize a highly performance web crawler so that structured records could be extracted efficiently from The Edge Malaysia, which consists of a variety of articles. This project focuses on the high performance-computing techniques such as multithreading, multiprocessing, and distributed computing to enhance the performance of the data processing pipeline.

1.2 Objectives

1. Develop a high-performance crawler to extract 100,000 structured records from a Malaysian website.
2. Process and clean the extracted data to prepare them for further analysis and storage.
3. Optimize the data processing pipeline by using multithreading, multiprocessing and distributed processing to increase speed and efficiency.
4. Evaluate the system's performance before and after optimization by comparing the key metrics like CPU usage, memory usage and throughput.

1.3 Target Website

The target website for the project is The Edge Malaysia which is a popular business and financial news website in Malaysia. https://theedgemalaysia.com/

1.4 Data to Be Extracted

The articles under the "Malaysia" section in the website are chosen. The table below outlines the attributes collected and their descriptions:

Table 1: Attributes collected and their descriptions

Attribute	Description
Category	Main domain of the article
Sub-category	Specific classification under the main category
Title	The headline the news article
Author	Name(s) of the person(s) who wrote the article
Source	Origin of the ariticle
Summary	A brief summary on the article's content
Created date	The original publication date of the article
Updated date	The date when the article was last updated

2.0 System Design and Architecture

2.1 Description of Architecture

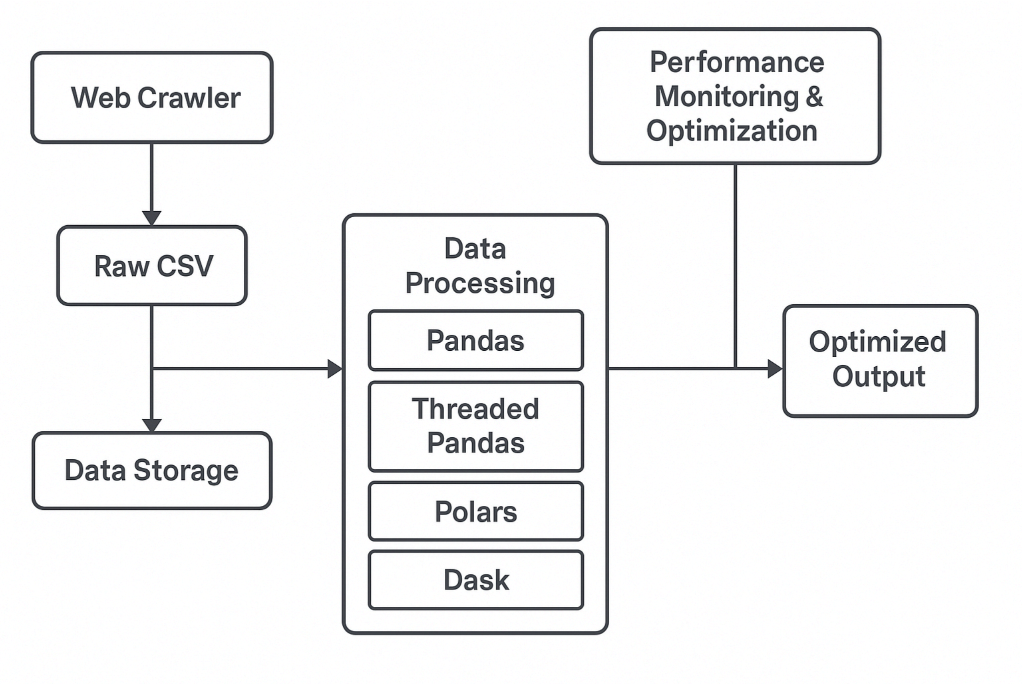


Figure 1: System architecture of the crawler

This component communicates directly with The Edge Malaysia's public API using Python's requests package. From the endpoint, it retrieves JSON responses: By increasing the offset option, the crawler manages pagination and gathers batches of ten items for each request. It consists of:

- Rate-limiting (delayed by 0.3 s) to accommodate server load.
- handling of unsuccessful request errors.
- Progressive saving: To guarantee data permanence in the event of a failure, backup CSV files are stored for every 10,000 articles.

Layer of Data Storage

CSV files with UTF-8 encoding are used to store the data. Following each scraping stage, intermediate files (such as theedge_articles.csv) are produced. Several versions of cleaned datasets are stored for comparison:

- theedge_cleaned_pandas.csv
- theedge_cleaned_polars.csv
- theedge_cleaned_dask.csv
- theedge_cleaned_pandas_threaded.csv

Data Processing

After data has been collected, it is run through a multi-functional cleaning engine that has been developed utilizing four different frameworks:

- Pandas standard for baseline
- Pandas with ThreadPoolExecutor for threaded processing
- Polars for fast, one-threading
- Dask for out-of-core and parallelizable (whether through the number of workers or additional cores) computations

Some of the cleaning functions include:

- Removing partial and full duplicates
- Filling in the empty spaces
- Removing whitespace
- Turning date strings into datetime objects
- Optimizing based types (categoricals for example)
- Performance Monitoring and Optimization

Every run has a monitoring thread wrapped in that records:

- Memory usage (peak and average)
- CPU usage (peak and average)
- Throughput and processing time (records/sec)

This extra layer allows for thorough comparison of system performance against each of the frameworks.

2.2 Tools and Frameworks Used

- Python (Requests)
- AsyncIO
- Dask
- Pandas / CSV / JSON

2.3 Roles of Team Members

Phase	Task	Member
Planning	● Coordination and project design	Pyng Haw
Data Collection	● Develop and test the initial web scraper to collect NST headlines ● Run the scraper progressively to gather 100k records	Pyng Haw
Data Processing	● Clean and structure the dataset (remove duplicates, fix casing, prepare CSV)	Jie Sheng
Optimization	● Apply optimization techniques ● Record performance metrics (time, memory, CPU usage)	Danial
Benchmarking	● Compile and analyze results, charts and graphs	Sabrina
Report Writing	Final documentation	All

3.0 Data Collection

3.1 Crawling Method

In this project, web crawling was done by directly hitting The Edge Malaysia's API as opposed to scraping content page-wise with tools such as Selenium or BeautifulSoup. This made it possible to fetch structured data smoothly and cost-effectively instead of sifting through complex HTML or interacting with dynamically loaded web pages.

Prior to trying this method, we had come across resources such as the Guide to Web Scraping APIs by Zyte explaining the advantages of scraping via APIs, besides watching a YouTube tutorial by John Watson Rooney to understand the implementation of API-based scraping. These resources made me realize the benefits that come with using APIs, such as faster data retrieval, lowered load on servers, and bypassing the issue of JavaScript-rendered content.

The project saw a data scraping method with Python's requests library utilized, for interaction with The Edge Malaysia's API directly. This then allowed us to glide past HTML parsing traditionally performed and grab article data instead in JSON format, which is way more reliable and efficient when it comes to massive scraping.

The scraping comprised the Malaysia news category by sending GET requests to the following API endpoint:

https://theedgemalaysia.com/api/loadMoreCategories?offset={offset}&categories=malaysia

The crawler, designed to solve pagination using an offset parameter that increased by 10 in every request, could handle 10 articles being returned in each response. To extract a big dataset, we iterated overrange of offsets to continuously collect articles in batches.

A simple rate-limiting method was used, with a short `time.sleep(0.3)` delay between requests, to let the server relax, reduce the possibility of a ban, and being polite to the server. Further, error handling was implemented in response to request failure, and timestamp conversion was performed to ensure the date fields (created date and updated date) were correctly generated for analysis.

To handle large amounts of data, the scraper was set up to store data automatically in chunks of 10,000 articles per CSV file. So if anything interrupted a scraping round, the partial data would still be available in small partitions as a backup.

Also, a final step was added to save the remaining articles, which would not fill a complete batch of 10,000 articles, in a separate CSV file, so none of the data would be lost.

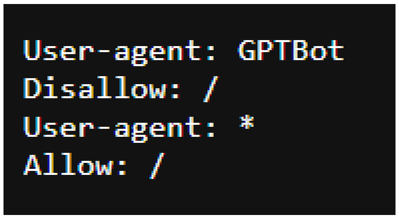
3.2 Number of Records Collected

The crawler successfully extracted 110,000 records.

3.3 Ethical Considerations

During the data collection process, we make sure the crawler follow the guidelines below:

- Respect the robots.txt file, where GPTBot is not allowed and All other bots are allowed as shown in Figure 1.
- Prevent excessive load on the target website by setting time interval between each scraping
- Only scrape publicly available content and avoid any private or restricted areas of the website
- Ensure that no personal data or sensitive information was collected during the scraping process



```
User-agent: GPTBot
Disallow: /
User-agent: *
Allow: /
```

Figure 2: The Edge Malaysia’s robots.txt

4.0 Data Processing

4.1 Cleaning Methods

a. Addressing Duplicate Data

Some articles in the news feed appeared many times because they were associated with several tags or categorizations. We removed duplicates so that each article would only appear as a single article, and made duplicates according to article title given that this should be a unique identifier.

```
df.drop_duplicates(subset='title', inplace=True)
```

b. Handle Missing Data

In order to safeguard the integrity and consistency of data, we took specific actions as follows:

1. Deleted the rows with no summary, because the summary column is essential to understanding the article's content.
2. Filled in missing values for the author and source column with "Unknown" in order to keep the record complete while not losing useful articles.

```
df = df[df['summary'].notna()]

df['author'].fillna('Unknown', inplace=True)

df['source'].fillna('Unknown', inplace=True)
```

4.2 Data Structure

The original raw dataset was scraped and saved as a CSV;

```
df = pd.read_csv('theedge_articles.csv')
```

After a series of cleaning and formatting, the resulting dataset was saved as a cleaned CSV;

```
df.to_csv('theedge_articles_cleaned.csv', index=False)
```

4.3 Transformation and Formatting

a. Date Conversion

To ensure the ability to sort properly and/or investigate the timeline of events, the created date and updated date data was allocated from string format to datetime format:

```
df['created date'] = pd.to_datetime(df['created date'], errors='coerce')
```

```
df['updated date'] = pd.to_datetime(df['updated date'], errors='coerce')
```

b. Category Structuring
The category column had multiple comma delimited labels. To enhance clarity and create more structured grouping, this field was split into three categories:

```
df[['category_1', 'category_2', 'category_3']] = df['category'].str.split(',', n=2, expand=True)
```

c. Additional Formatting
While this dataset did not have a Place column as in the example provided, we applied some standard word processing formatting principles across the textual data to normalize it, for example, fixing excessive whitespace on categories and titles, and ensuring that all values were consistently cased (where applicable).

5.0 Optimization Techniques

5.1 Methods Used

1. multithreading - pandas with threading

- Purpose: Increase the speed of I/O bound processes like reading several CSV files or using functions across Dataframes at times
- Approach: Multiple **threads** can be launched for data ingestion or processing tasks using Python’s built-in threading module
- Benefit:Boosts performance on systems with I/O bottlenecks,although Python’s Global Interpreter Lock (GIL) limits CPU-bound operations.

2. Distributed processing - Dask

- Purpose: Expands pandas-like functions to several machines or cores.
- Approach: Dask is used. dataframe that mimics the pandas API but divides the workload among partitions and performs calculations sluggishly.
- Benefit: is capable of handling datasets bigger than memory and provides parallel computation through the use of distributed clusters or multiple threads.

3. library - Polars

- Purpose: uses Rust under the hood to carry out DataFrame operations more effectively than Pandas.
- Approach: By default, it uses multithreaded execution and lazy evaluation.
- Benefit: reduced memory footprint and faster execution time, particularly useful for filtering/aggregation and columnar operations.

5.2 Code Overview

Multithreading - pandas with threading

```
Python
import pandas as pd
import time
import psutil
import threading
from concurrent.futures import ThreadPoolExecutor

# Global flag to stop monitoring
monitoring = True

# Function to monitor performance DURING the process
def monitor_performance(log_list, interval=1):
    process = psutil.Process()
    while monitoring:
        mem_usage = process.memory_info().rss / (1024 * 1024)  # MB
        cpu_usage = process.cpu_percent(interval=None)         # %
        log_list.append((time.time(), mem_usage, cpu_usage))
        time.sleep(interval)

# List to store performance logs
performance_logs = []

# Start monitoring in a separate thread
monitor_thread = threading.Thread(target=monitor_performance,
args=(performance_logs, 0.5))
monitor_thread.start()

# Track time
start_time = time.time()

print("==============================")
print("🔄 Starting cleaning process (fully threaded column
cleaning)...")
print("==============================")

# 📥 Load the dataset
df = pd.read_csv('theedge_articles.csv', low_memory=False)
```

```
print(f"✅ Loaded {len(df)} records.")

# 🔍 Check for missing values
print("Missing values per column:")
print(df.isnull().sum())


# 🧹 Fill missing values (threaded)
print("✅ Filling missing values (threaded)...")

def fill_missing(col_value):
    col, value = col_value
    df[col] = df[col].fillna(value)

fill_values = {
    'sub-category': 'General',
    'author': 'Unknown',
    'source': 'Unknown',
    'summary': 'N/A',
    'updated date': 'NaT'
}

with ThreadPoolExecutor(max_workers=len(fill_values)) as executor:
    executor.map(fill_missing, fill_values.items())

# 🗑️ Remove exact duplicates
initial_len = len(df)
df = df.drop_duplicates().copy()
print(f"Removed {initial_len - len(df)} exact duplicate records.")

# 🗑️ Remove duplicates based on 'title'
initial_len = len(df)
df = df.drop_duplicates(subset=['title']).copy()
print(f"Removed {initial_len - len(df)} duplicate records based on
title.")

# 🕒 Convert date columns to datetime (threaded)
print("✅ Converting date columns to datetime (threaded)")

def convert_date(col):
    df[col] = pd.to_datetime(df[col], errors='coerce')

date_cols = ['created date', 'updated date']

with ThreadPoolExecutor(max_workers=len(date_cols)) as executor:
    executor.map(convert_date, date_cols)

# ✂️ Strip whitespace from text columns (threaded)
print("✅ Stripping whitespace from text columns (threaded)")

def strip_column(col):
    df[col] = df[col].astype(str).str.strip()

text_cols = ['title', 'summary', 'category', 'sub-category', 'author',
'source']

with ThreadPoolExecutor(max_workers=len(text_cols)) as executor:
    executor.map(strip_column, text_cols)

# 💾 Save cleaned data
df.to_csv('theedge_cleaned_pandas_threaded.csv', index=False,
encoding='utf-8-sig', na_rep='N/A')
print(f"✅ Saved cleaned dataset: {len(df)} records to
'theedge_cleaned_pandas_threaded.csv'.")

# Stop monitoring
monitoring = False
monitor_thread.join()

end_time = time.time()

total_time = end_time - start_time
num_records = df.shape[0]
throughput = num_records / total_time

# ==========================
# 🚀 Performance Summary
# ==========================
print("==============================")
print("🚀 Performance Summary")
print("==============================")
print(f"Total time taken: {total_time:.2f} seconds")

# Print memory & CPU usage
peak_mem = max([m for _, m, _ in performance_logs])
peak_cpu = max([c for _, _, c in performance_logs])
avg_mem = sum([m for _, m, _ in performance_logs]) /
len(performance_logs)
avg_cpu = sum([c for _, _, c in performance_logs]) /
len(performance_logs)
print(f"Average memory usage during process: {avg_mem:.2f} MB")
print(f"Peak memory usage during process: {peak_mem:.2f} MB")
print(f"Average CPU usage during process: {avg_cpu:.2f}%")
print(f"Peak CPU usage during process: {peak_cpu:.2f}%")
print(f"✅ Processed {num_records} records in {total_time:.2f}
seconds.")
print(f"🚀 Throughput: {throughput:.2f} records per second.")
```

Distributed processing - Dask

Python

```
import dask.dataframe as dd
import time
import psutil
import threading

# Global flag to stop monitoring
monitoring = True

def monitor_performance(log_list, interval=1):
    process = psutil.Process()
    while monitoring:
        mem_usage = process.memory_info().rss / (1024 * 1024)  # MB
        cpu_usage = process.cpu_percent(interval=None)
        log_list.append((time.time(), mem_usage, cpu_usage))
        time.sleep(interval)

performance_logs = []
monitor_thread      =       threading.Thread(target=monitor_performance,
args=(performance_logs, 0.5))
monitor_thread.start()

start_time = time.time()
print("==============================")
print("🔄 Starting Dask cleaning process...")
print("==============================")

# ✅ Tell Dask the exact type of each column
ddf = dd.read_csv('theedge_articles.csv', low_memory=False)
print(f"✅ Loaded {len(ddf)} records.")

# Cleaning steps (same as before)
print("✅ Filling missing values...")
ddf['sub-category'] = ddf['sub-category'].fillna('General')
ddf['author'] = ddf['author'].fillna('Unknown')
ddf['source'] = ddf['source'].fillna('Unknown')
ddf['summary'] = ddf['summary'].fillna('')
ddf['updated date'] = ddf['updated date'].fillna('NaT')
```

```
# Remove duplicates
ddf = ddf.drop_duplicates().drop_duplicates(subset=['title'])
print("✅ Removed duplicates.")

# Strip whitespace
text_cols = ['title', 'summary', 'category', 'sub-category', 'author',
'source']
for col in text_cols:
    if col in ddf.columns:
        ddf[col] = ddf[col].astype(str).str.strip()
print("✅ Stripped whitespace from text columns")

# Save result
ddf.compute().to_csv('theedge_cleaned_dask.csv', index=False,
encoding='utf-8-sig', na_rep='N/A')
print("✅ Saved cleaned dataset to 'theedge_cleaned_dask.csv'.")

# Stop monitoring
monitoring = False
monitor_thread.join()

end_time = time.time()

total_time = end_time - start_time
num_records = ddf.shape[0].compute()
throughput = num_records / total_time

print("==============================")
print("🚀 Performance Summary")
print("==============================")
print(f"Total time taken: {end_time - start_time:.2f} seconds")

# Peak + average usage
peak_mem = max([m for _, m, _ in performance_logs])
peak_cpu = max([c for _, _, c in performance_logs])
avg_mem = sum([m for _, m, _ in performance_logs]) /
len(performance_logs)
avg_cpu = sum([c for _, _, c in performance_logs]) /
len(performance_logs)
```

```python
print(f"Average memory: {avg_mem:.2f} MB")
print(f"Peak memory: {peak_mem:.2f} MB")
print(f"Average CPU: {avg_cpu:.2f}%")
print(f"Peak CPU: {peak_cpu:.2f}%")
print(f"✅ Processed {num_records} records in {total_time:.2f} seconds.")
print(f"🚀 Throughput: {throughput:.2f} records per second.")
print("==============================")
```

Library - Polars

Python
```python
import polars as pl
import time
import psutil
import threading

# Global flag to stop monitoring
monitoring = True

# Function to monitor performance DURING the process
def monitor_performance(log_list, interval=1):
    process = psutil.Process()
    while monitoring:
        mem_usage = process.memory_info().rss / (1024 * 1024)  # MB
        cpu_usage = process.cpu_percent(interval=None)         # %
        log_list.append((time.time(), mem_usage, cpu_usage))
        time.sleep(interval)


# List to store performance logs
performance_logs = []


# Start monitoring in a separate thread
monitor_thread      =      threading.Thread(target=monitor_performance,
args=(performance_logs, 0.5))
monitor_thread.start()

# Track time
start_time = time.time()

print("==============================")
print("🔄 Starting Polars cleaning process...")
print("==============================")

# 📥 Load the dataset
df = pl.read_csv('theedge_articles.csv', low_memory=False)
print(f"✅ Loaded {df.shape[0]} records.")

# 🔍 Check for missing values
```

```
null_counts = df.null_count()
print("Missing values per column:")
print(null_counts)

# 🧹 Fill missing values
fill_values = {
    'sub-category': 'General',
    'author': 'Unknown',
    'source': 'Unknown',
    'summary': '',
    'updated date': 'NaT'
}
for column, value in fill_values.items():
    if column in df.columns:
        df = df.with_columns(
            pl.col(column).fill_null(value)
        )
print("✅ Filled missing values.")

# 🗑️ Remove exact duplicates
initial_len = df.shape[0]
df = df.unique()
print(f"Removed {initial_len - df.shape[0]} exact duplicate records.")


# 🗑️ Remove duplicates based on 'title'
initial_len = df.shape[0]
if 'title' in df.columns:
    df = df.unique(subset=['title'])
    print(f"Removed {initial_len - df.shape[0]} duplicate records based
on title.")
else:
      print("⚠️ 'title' column not found; skipping duplicate removal
based on title.")


# 🕒 Convert date columns to datetime
if 'created date' in df.columns:
    df = df.with_columns(
                                                         pl.col('created
date').str.strip_chars().str.strptime(pl.Datetime,
strict=False).alias('created date')
    )
if 'updated date' in df.columns:
    df = df.with_columns(
                                                         pl.col('updated
date').str.strip_chars().str.strptime(pl.Datetime,
strict=False).alias('updated date')
    )
print("✅ Converted date columns to datetime.")


# ✂️ Strip whitespace from text columns
text_cols = ['title', 'summary', 'category', 'sub-category', 'author',
'source']
for col in text_cols:
    if col in df.columns:
        df = df.with_columns(
            pl.col(col).cast(pl.Utf8).str.strip_chars()
        )
print("✅ Stripped whitespace from text columns.")


# 💾 Save cleaned data
df.write_csv('theedge_cleaned_polars.csv')
print(f"✅ Saved cleaned dataset: {df.shape[0]} records to
'theedge_cleaned_polars.csv'.")

# Stop monitoring
monitoring = False
monitor_thread.join()

end_time = time.time()

total_time = end_time - start_time
num_records = df.shape[0]
throughput = num_records / total_time

# ==========================
# 🚀 Performance Summary
# ==========================
print("==============================")
print("🚀 Performance Summary")
print("==============================")
print(f"Total time taken: {end_time - start_time:.2f} seconds")


# Peak + average usage
peak_mem = max([m for _, m, _ in performance_logs])
peak_cpu = max([c for _, _, c in performance_logs])
avg_mem = sum([m for _, m, _ in performance_logs]) /
len(performance_logs)
avg_cpu = sum([c for _, _, c in performance_logs]) /
len(performance_logs)
print(f"Average memory usage during process: {avg_mem:.2f} MB")
print(f"Peak memory usage during process: {peak_mem:.2f} MB")
print(f"Average CPU usage during process: {avg_cpu:.2f}%")
print(f"Peak CPU usage during process: {peak_cpu:.2f}%")
print(f"✅ Processed {num_records} records in {total_time:.2f}
seconds.")
print(f"🚀 Throughput: {throughput:.2f} records per second.")
print("==============================")
```

6.0 Performance Evaluation

6.1 Time Taken in seconds

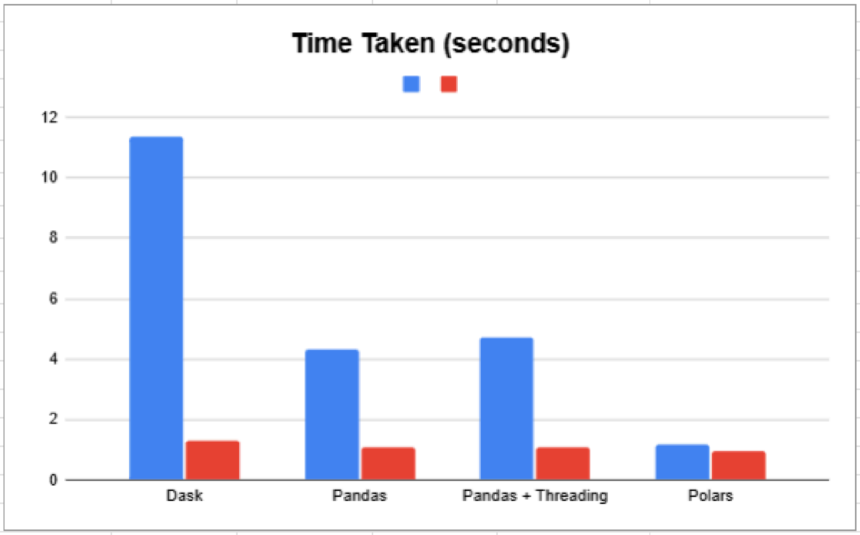


Figure 3: Comparison of Time Taken (in seconds) across Different Libraries

Library	Before Optimization (s)	After Optimization (s)
Pandas	4.31	1.07
Pandas + Threading	4.70	1.08
Dask	11.37	1.32
Polars	1.17	0.96

All libraries demonstrated a significant reduction in processing time after optimization, with Dask and Polars spotting the biggest decreases (from 11.37 to 1.32 seconds and 1.17 to 0.96 seconds, respectively). Polars was always the fastest both before and after optimization. Though not as significantly, pandas and pandas with threading also improved. This shows that Polars is extremely efficient, and Dask performs much better with optimization, even though it's the worst performer without optimization.

6.2 CPU Usage (%)

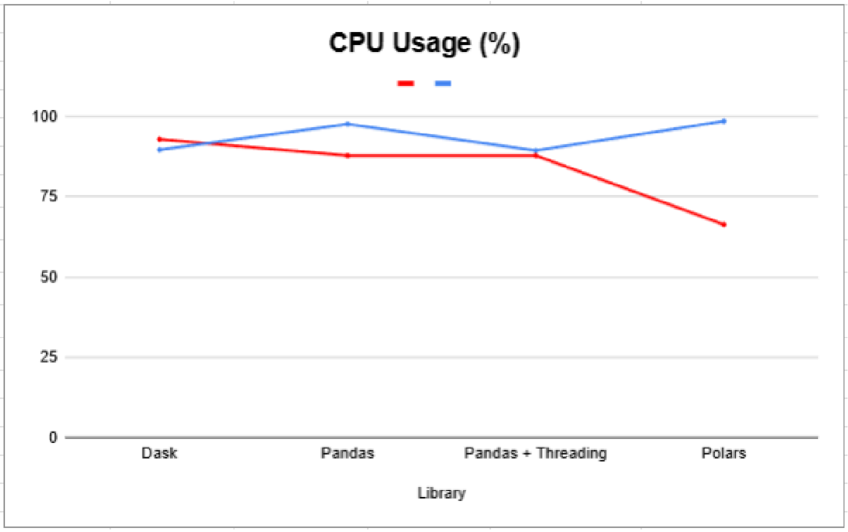


Figure 4: CPU Usage (%) for Various Libraries during Data Processing

Library	Before Optimization (%)	After Optimization (%)
Pandas	87.90	97.71
Pandas + Threading	87.91	89.47
Dask	92.97	89.71
Polars	66.39	98.60

After optimization, CPU usage generally went up, particularly for Pandas (from 87.90% to 97.71%) and Polars (from 66.39% to 98.60%). This suggests that the optimized processes used CPU resources more efficiently, most likely as a result of vectorized or parallelized operations. A more balanced or effective execution is suggested by the small decrease in CPU usage for Dask and Pandas + Threading.

6.3 Memory Usage (MB)

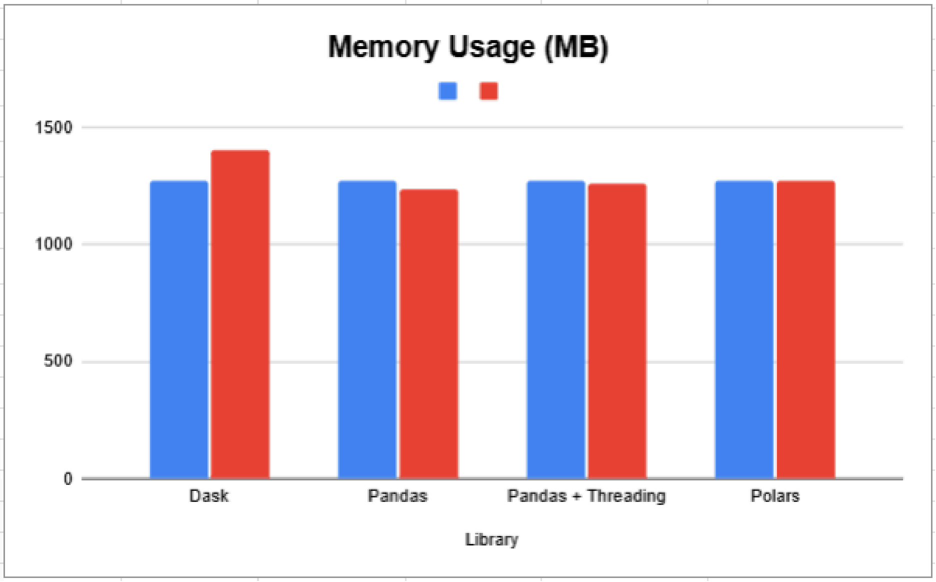


Figure 5: Memory Usage (MB) by Each Library

Library	Before Optimization (MB)	After Optimization (MB)
Pandas	1237.40	1275.09
Pandas + Threading	1263.07	1274.09
Dask	1407.42	1275.30
Polars	1275.69	1273.69

There were only minor variations in memory usage between libraries before and after optimization.Prior to optimization, Dask used the most memory; following optimization, this usage was slightly reduced. Pandas and Polars demonstrated good memory efficiency by staying within a similar range. Overall, there was little increase in memory usage, indicating that optimization worked well without requiring significant memory trade-offs.

6.4 Throughput (records/s)

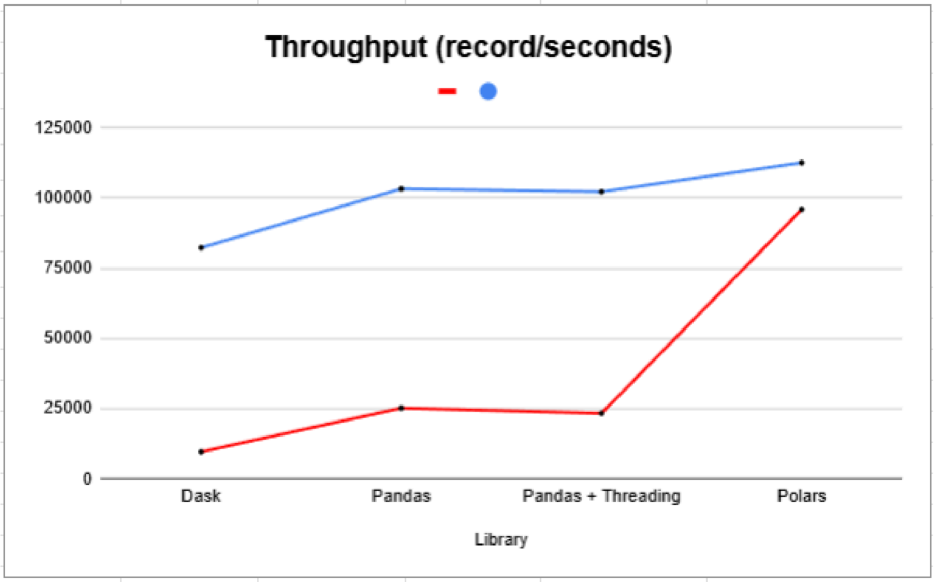


Figure 6: Throughput (record/s) Comparison across Libraries

Library	Before Optimization (record/s)	After Optimization (record/s)
Pandas	25115.66	103236.63
Pandas + Threading	23255.58	102188.66
Dask	9528.43	82263.68
Polars	95919.64	112545.89

All libraries experienced a significant increase in throughput, but Dask and Polars saw the biggest increases, going from 9,528 to 82,263 records/s and 95,919 to 112,546 records/s, respectively. This demonstrates how optimization resulted in quicker data processing. Once more, Polars performed better than the others, followed by Pandas and Pandas + Threading, proving that Polars is the best option for tasks requiring a lot of performance.

7.0 Challenges and Limitations

7.1 Challenges

1. Dynamic content loading via JavaScript

    a. The number of target data contained in JavaScript-processed data. Traditional scraping methods (requests, bs4) didn’t cut it, as those libs do not process JS. This made it necessary to resort to other more sophisticated tools such as Selenium or headless browsers (Puppeteer) which slowed down even further the scraping process and the computer resources’ use.

2. IP blocking due to aggressive crawling

    a. Heavy request traffic with high frequency and volume activated antiscraping on the host website, including frequent judging, IP block, and captcha verification. This was a significant barrier to data collection. To circumvent this, rate-limiting etc. functions as user-agent rotation and time intervals were employed to simulate human like activities. But these solutions compromised the overall acquisition speed.

3. Data inconsistency across pages

    a. Data format and structure differed from one web page to another, making the parsing logic more difficult to understand. Fields were occasionally absent or incorrectly labeled, necessitating the use of strong exception handling and normalization protocols throughout the data transformation and cleaning process.

7.2 Limitations

1. Site structure may change, breaking the crawler

    a. The logic for data extraction was overly bound to the particular configuration of the target website. Modifications to the HTML structure or the class labels could disable the crawler which would need updates to the scraping scripts.

2. Performance limited by local machine resources

    a. The hardware constraints of the local machine ultimately limited the system's overall performance, even though tools like Dask and Polars increased processing efficiency. Large dataset processing or the rendering of pages with a lot of JavaScript put a heavy strain on CPU and memory, which limits scalability.

8.0 Conclusion and Future Work

8.1 Summary of Findings

The test performance shows that the optimized crawler substantially enhanced every metrics that was tested. The processing time was drastically decreased and CPU and memory usage were managed by the system more efficiently. Throughput considerably better with Polars getting the fastest speed, then followed by Pandas and Pandas with threading. Even though Dask is slower in this case, it is still relevant for larger-scale distributed tasks. Overall, the crawler successfully collected and processed over 100,000 records, showing the efficiency and the scalability of optimization.

8.2 Future Work

A few improvements can be taken into consideration when planning for upcoming upgrades. The crawler's reliability can be enhanced by including retry mechanisms and proxy rotation, especially when managing unpredictable connections or preventing IP blocking. Gathering and storing the collected data straight into a cloud-hosted database would streamline the access and real-time data analysis would be supported. Furthermore, allowing the crawler to gather data across sites would improve the datasets and enhance its potential for more general uses like trend analysis, sentiment monitoring and comparative studies.

9.0 References

John Watson Rooney. (2021, August 1). *Always Check for the Hidden API when Web Scraping* [Video]. YouTube. https://www.youtube.com/watch?v=DqtlR0y0suo

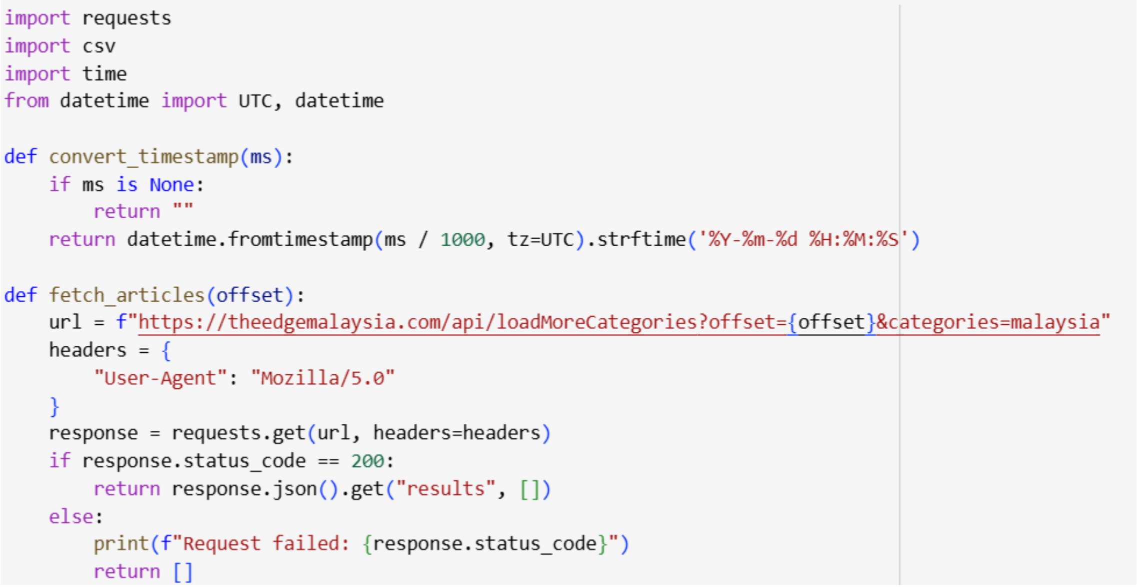
Guide to Web Scraping APIs: Key features and Benefits. (n.d.). https://www.zyte.com/learn/guide-to-web-scraping-ap-is-key-features-and-benefits/#how-web-scraping-apis-work

Noorani, W. (2024, November 15). Python for Big Data: Essential libraries and effective strategies for advanced data science. *Medium*. https://medium.com/data-and-beyond/python-for-big-data-essential-libraries-and-effective-strategies-for-advanced-data-science-02efe71063b4#df88

Python Libraries for Data Clean-Up - StrataScratch. (2024, September 11). Stratascratch.com. https://www.stratascratch.com/blog/python-libraries-for-data-clean-up/

10.0 Appendices

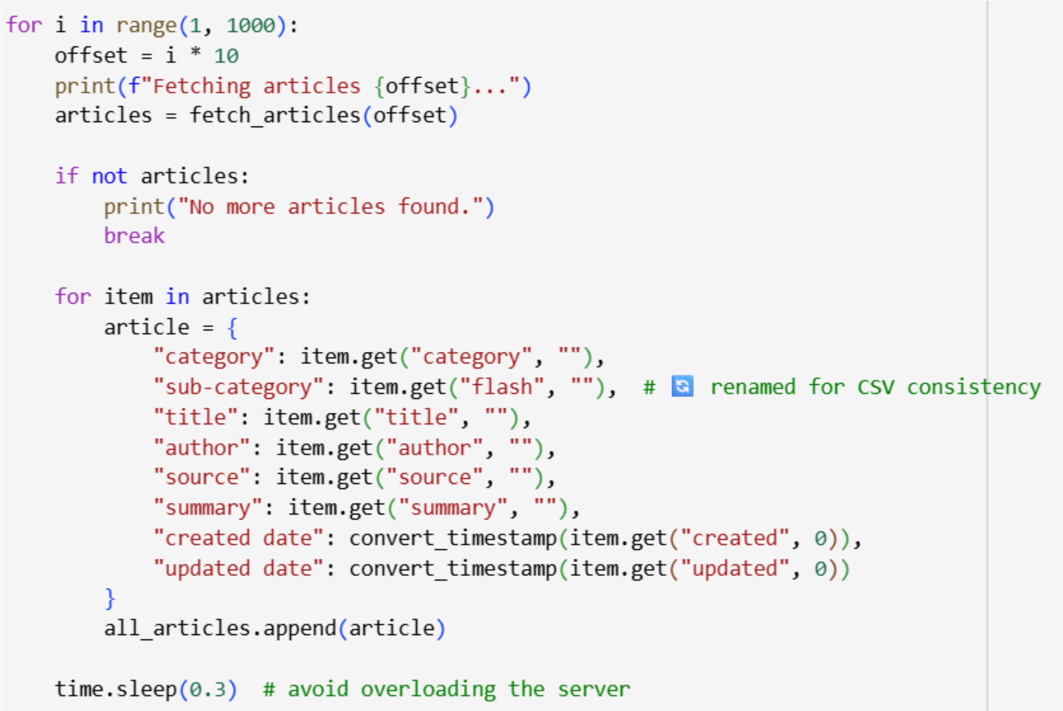
10.1 Sample Code Snippets



```python
import requests
import csv
import time
from datetime import UTC, datetime

def convert_timestamp(ms):
    if ms is None:
        return ""
    return datetime.fromtimestamp(ms / 1000, tz=UTC).strftime('%Y-%m-%d %H:%M:%S')

def fetch_articles(offset):
    url = f"https://theedgemalaysia.com/api/loadMoreCategories?offset={offset}&categories=malaysia"
    headers = {
        "User-Agent": "Mozilla/5.0"
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json().get("results", [])
    else:
        print(f"Request failed: {response.status_code}")
        return []
```

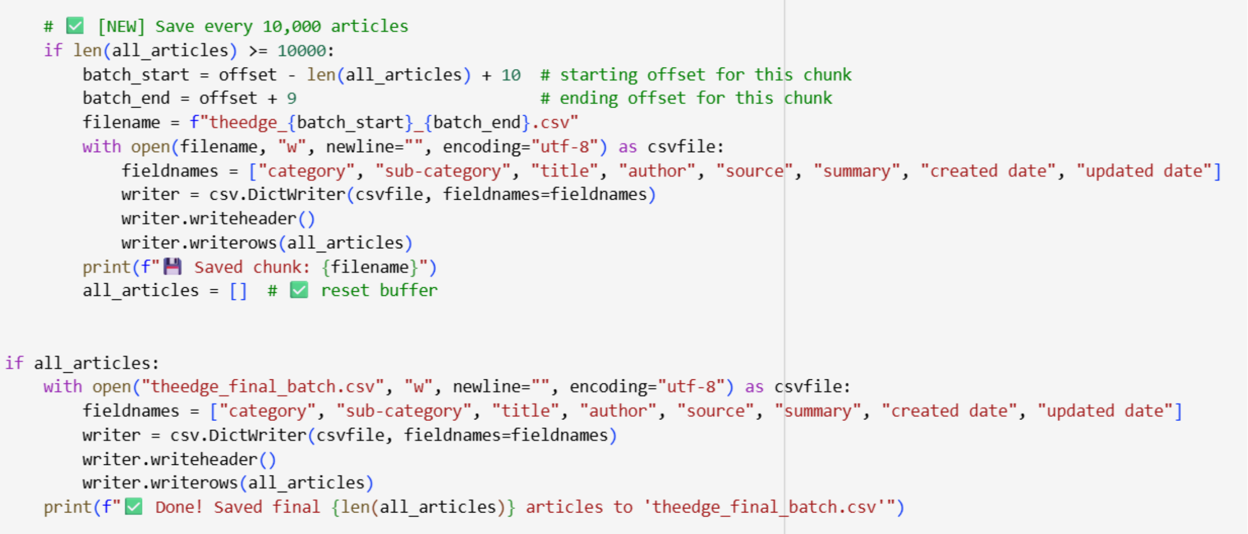


```python
for i in range(1, 1000):
    offset = i * 10
    print(f"Fetching articles {offset}...")
    articles = fetch_articles(offset)

    if not articles:
        print("No more articles found.")
        break

    for item in articles:
        article = {
            "category": item.get("category", ""),
            "sub-category": item.get("flash", ""),  # 🔄 renamed for CSV consistency
            "title": item.get("title", ""),
            "author": item.get("author", ""),
            "source": item.get("source", ""),
            "summary": item.get("summary", ""),
            "created date": convert_timestamp(item.get("created", 0)),
            "updated date": convert_timestamp(item.get("updated", 0))
        }
        all_articles.append(article)

    time.sleep(0.3)  # avoid overloading the server
```

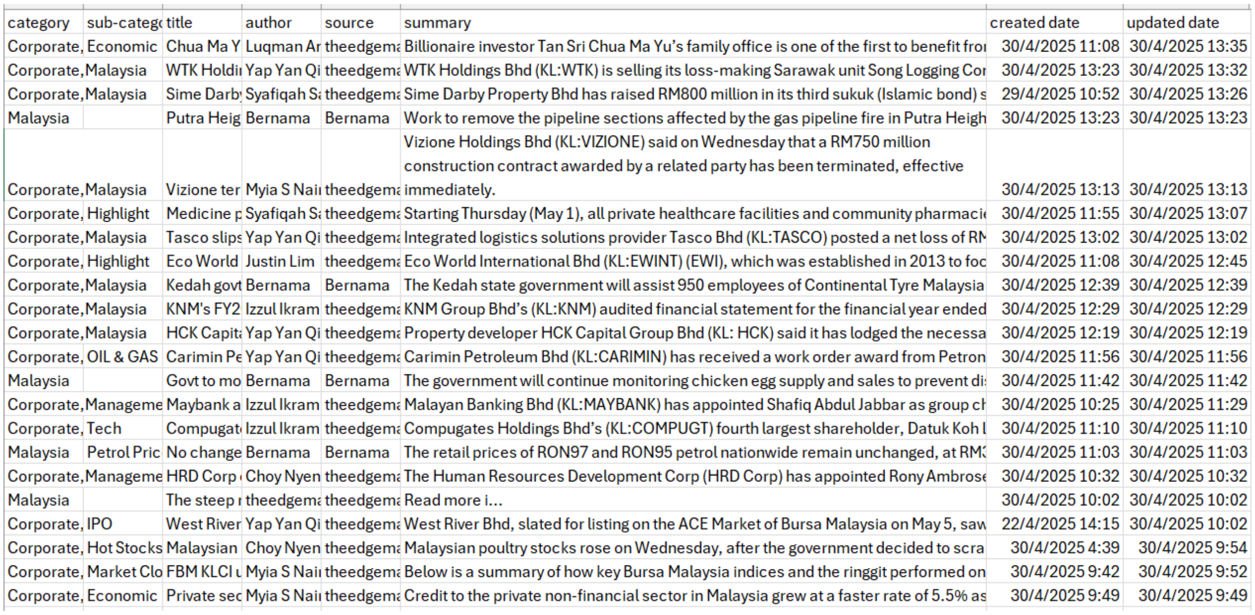


```
    # ✅ [NEW] Save every 10,000 articles
    if len(all_articles) >= 10000:
        batch_start = offset - len(all_articles) + 10  # starting offset for this chunk
        batch_end = offset + 9                         # ending offset for this chunk
        filename = f"theedge_{batch_start}_{batch_end}.csv"
        with open(filename, "w", newline="", encoding="utf-8") as csvfile:
            fieldnames = ["category", "sub-category", "title", "author", "source", "summary", "created date", "updated date"]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(all_articles)
        print(f"💾 Saved chunk: {filename}")
        all_articles = []  # ✅ reset buffer


if all_articles:
    with open("theedge_final_batch.csv", "w", newline="", encoding="utf-8") as csvfile:
        fieldnames = ["category", "sub-category", "title", "author", "source", "summary", "created date", "updated date"]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(all_articles)
    print(f"✅ Done! Saved final {len(all_articles)} articles to 'theedge_final_batch.csv'")
```

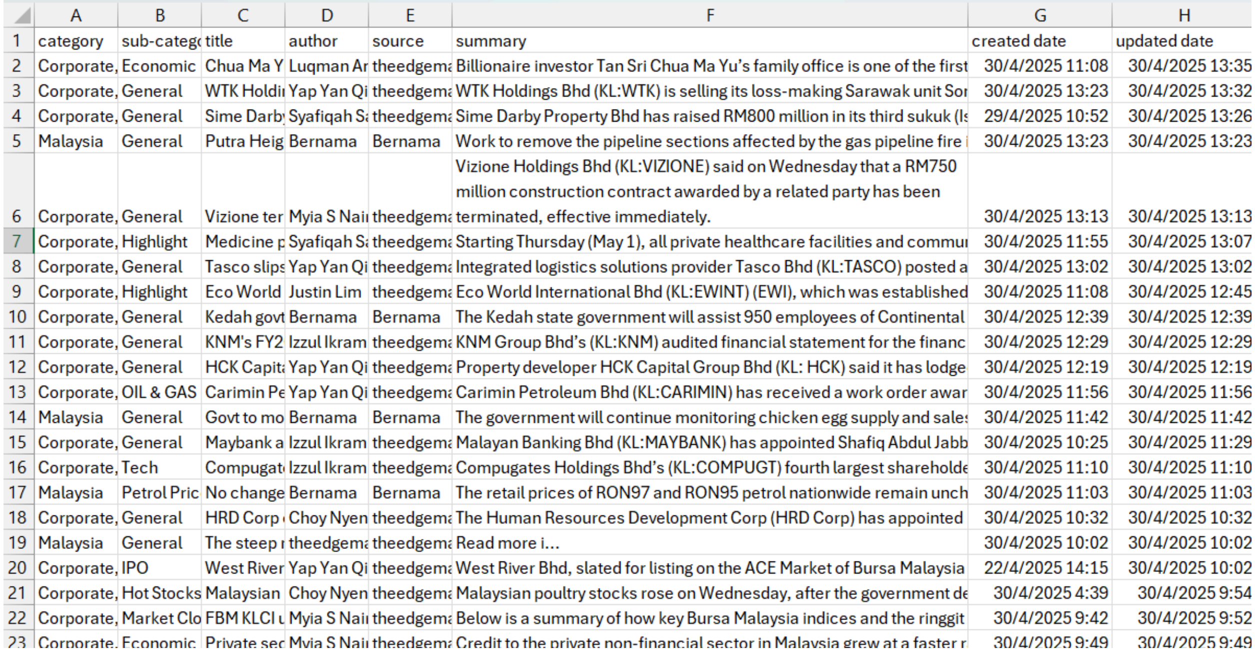
Figure 7: Main Crawler Code

10.2 Screenshots of Output



category	sub-categ	title	author	source	summary	created date	updated date
Corporate,	Economic	Chua Ma Y	Luqman Ar	theedgema	Billionaire investor Tan Sri Chua Ma Yu's family office is one of the first to benefit fro	30/4/2025 11:08	30/4/2025 13:35
Corporate,	Malaysia	WTK Holdi	Yap Yan Qi	theedgema	WTK Holdings Bhd (KL:WTK) is selling its loss-making Sarawak unit Song Logging Cor	30/4/2025 13:23	30/4/2025 13:32
Corporate,	Malaysia	Sime Darb	Syafiqah Sa	theedgema	Sime Darby Property Bhd has raised RM800 million in its third sukuk (Islamic bond) s	29/4/2025 10:52	30/4/2025 13:26
Malaysia		Putra Heig	Bernama	Bernama	Work to remove the pipeline sections affected by the gas pipeline fire in Putra Heigh	30/4/2025 13:23	30/4/2025 13:23
Corporate,	Malaysia	Vizione ter	Myia S Nai	theedgema	Vizione Holdings Bhd (KL:VIZIONE) said on Wednesday that a RM750 million construction contract awarded by a related party has been terminated, effective immediately.	30/4/2025 13:13	30/4/2025 13:13
Corporate,	Highlight	Medicine p	Syafiqah Sa	theedgema	Starting Thursday (May 1), all private healthcare facilities and community pharmaci	30/4/2025 11:55	30/4/2025 13:07
Corporate,	Malaysia	Tasco slips	Yap Yan Qi	theedgema	Integrated logistics solutions provider Tasco Bhd (KL:TASCO) posted a net loss of RM	30/4/2025 13:02	30/4/2025 13:02
Corporate,	Highlight	Eco World	Justin Lim	theedgema	Eco World International Bhd (KL:EWINT) (EWI), which was established in 2013 to foc	30/4/2025 11:08	30/4/2025 12:45
Corporate,	Malaysia	Kedah govt	Bernama	Bernama	The Kedah state government will assist 950 employees of Continental Tyre Malaysia	30/4/2025 12:39	30/4/2025 12:39
Corporate,	Malaysia	KNM's FY2	Izzul Ikram	theedgema	KNM Group Bhd's (KL:KNM) audited financial statement for the financial year ended	30/4/2025 12:29	30/4/2025 12:29
Corporate,	Malaysia	HCK Capita	Yap Yan Qi	theedgema	Property developer HCK Capital Group Bhd (KL: HCK) said it has lodged the necessa	30/4/2025 12:19	30/4/2025 12:19
Corporate,	OIL & GAS	Carimin Pe	Yap Yan Qi	theedgema	Carimin Petroleum Bhd (KL:CARIMIN) has received a work order award from Petron	30/4/2025 11:56	30/4/2025 11:56
Malaysia		Govt to mo	Bernama	Bernama	The government will continue monitoring chicken egg supply and sales to prevent di	30/4/2025 11:42	30/4/2025 11:42
Corporate,	Manageme	Maybank a	Izzul Ikram	theedgema	Malayan Banking Bhd (KL:MAYBANK) has appointed Shafiq Abdul Jabbar as group ch	30/4/2025 10:25	30/4/2025 11:29
Corporate,	Tech	Compugat	Izzul Ikram	theedgema	Compugates Holdings Bhd's (KL:COMPUGT) fourth largest shareholder, Datuk Koh l	30/4/2025 11:10	30/4/2025 11:10
Malaysia	Petrol Pric	No change	Bernama	Bernama	The retail prices of RON97 and RON95 petrol nationwide remain unchanged, at RM3	30/4/2025 11:03	30/4/2025 11:03
Corporate,	Manageme	HRD Corp	Choy Nyen	theedgema	The Human Resources Development Corp (HRD Corp) has appointed Rony Ambrose	30/4/2025 10:32	30/4/2025 10:32
Malaysia		The steep r	theedgema	theedgema	Read more i…	30/4/2025 10:02	30/4/2025 10:02
Corporate,	IPO	West River	Yap Yan Qi	theedgema	West River Bhd, slated for listing on the ACE Market of Bursa Malaysia on May 5, saw	22/4/2025 14:15	30/4/2025 10:02
Corporate,	Hot Stocks	Malaysian	Choy Nyen	theedgema	Malaysian poultry stocks rose on Wednesday, after the government decided to scra	30/4/2025 4:39	30/4/2025 9:54
Corporate,	Market Clo	FBM KLCI u	Myia S Nai	theedgema	Below is a summary of how key Bursa Malaysia indices and the ringgit performed on	30/4/2025 9:42	30/4/2025 9:52
Corporate,	Economic	Private sec	Myia S Nai	theedgema	Credit to the private non-financial sector in Malaysia grew at a faster rate of 5.5% as	30/4/2025 9:49	30/4/2025 9:49

Figure 8: Raw Data



	A	B	C	D	E	F	G	H
1	category	sub-categ	title	author	source	summary	created date	updated date
2	Corporate,	Economic	Chua Ma Y	Luqman Ar	theedgema	Billionaire investor Tan Sri Chua Ma Yu's family office is one of the first	30/4/2025 11:08	30/4/2025 13:35
3	Corporate,	General	WTK Holdi	Yap Yan Qi	theedgema	WTK Holdings Bhd (KL:WTK) is selling its loss-making Sarawak unit Sor	30/4/2025 13:23	30/4/2025 13:32
4	Corporate,	General	Sime Darby	Syafiqah Sa	theedgema	Sime Darby Property Bhd has raised RM800 million in its third sukuk (Is	29/4/2025 10:52	30/4/2025 13:26
5	Malaysia	General	Putra Heig	Bernama	Bernama	Work to remove the pipeline sections affected by the gas pipeline fire i	30/4/2025 13:23	30/4/2025 13:23
6	Corporate,	General	Vizione ter	Myia S Nai	theedgema	Vizione Holdings Bhd (KL:VIZIONE) said on Wednesday that a RM750 million construction contract awarded by a related party has been terminated, effective immediately.	30/4/2025 13:13	30/4/2025 13:13
7	Corporate,	Highlight	Medicine p	Syafiqah Sa	theedgema	Starting Thursday (May 1), all private healthcare facilities and commu	30/4/2025 11:55	30/4/2025 13:07
8	Corporate,	General	Tasco slips	Yap Yan Qi	theedgema	Integrated logistics solutions provider Tasco Bhd (KL:TASCO) posted a	30/4/2025 13:02	30/4/2025 13:02
9	Corporate,	Highlight	Eco World	Justin Lim	theedgema	Eco World International Bhd (KL:EWINT) (EWI), which was established	30/4/2025 11:08	30/4/2025 12:45
10	Corporate,	General	Kedah govt	Bernama	Bernama	The Kedah state government will assist 950 employees of Continental	30/4/2025 12:39	30/4/2025 12:39
11	Corporate,	General	KNM's FY2	Izzul Ikram	theedgema	KNM Group Bhd's (KL:KNM) audited financial statement for the financ	30/4/2025 12:29	30/4/2025 12:29
12	Corporate,	General	HCK Capita	Yap Yan Qi	theedgema	Property developer HCK Capital Group Bhd (KL: HCK) said it has lodge	30/4/2025 12:19	30/4/2025 12:19
13	Corporate,	OIL & GAS	Carimin Pe	Yap Yan Qi	theedgema	Carimin Petroleum Bhd (KL:CARIMIN) has received a work order awar	30/4/2025 11:56	30/4/2025 11:56
14	Malaysia	General	Govt to mo	Bernama	Bernama	The government will continue monitoring chicken egg supply and sales	30/4/2025 11:42	30/4/2025 11:42
15	Corporate,	General	Maybank a	Izzul Ikram	theedgema	Malayan Banking Bhd (KL:MAYBANK) has appointed Shafiq Abdul Jabb	30/4/2025 10:25	30/4/2025 11:29
16	Corporate,	Tech	Compugat	Izzul Ikram	theedgema	Compugates Holdings Bhd's (KL:COMPUGT) fourth largest sharehold	30/4/2025 11:10	30/4/2025 11:10
17	Malaysia	Petrol Pric	No change	Bernama	Bernama	The retail prices of RON97 and RON95 petrol nationwide remain unch	30/4/2025 11:03	30/4/2025 11:03
18	Corporate,	General	HRD Corp	Choy Nyen	theedgema	The Human Resources Development Corp (HRD Corp) has appointed	30/4/2025 10:32	30/4/2025 10:32
19	Malaysia	General	The steep	theedgema	theedgema	Read more i...	30/4/2025 10:02	30/4/2025 10:02
20	Corporate,	IPO	West River	Yap Yan Qi	theedgema	West River Bhd, slated for listing on the ACE Market of Bursa Malaysia	22/4/2025 14:15	30/4/2025 10:02
21	Corporate,	Hot Stocks	Malaysian	Choy Nyen	theedgema	Malaysian poultry stocks rose on Wednesday, after the government de	30/4/2025 4:39	30/4/2025 9:54
22	Corporate,	Market Clo	FBM KLCI u	Myia S Nai	theedgema	Below is a summary of how key Bursa Malaysia indices and the ringgit	30/4/2025 9:42	30/4/2025 9:52
23	Corporate,	Economic	Private sec	Myia S Nai	theedgema	Credit to the private non-financial sector in Malaysia grew at a faster r	30/4/2025 9:49	30/4/2025 9:49

Figure 9: Cleaned data

10.3 Links to Full Code Repository and Dataset

- GitHub:

  https://github.com/drshahizan/HPDP/tree/main/2425/project/p1/GroupG/p1

- Dataset:

  https://github.com/drshahizan/HPDP/tree/main/2425/project/p1/GroupG/data