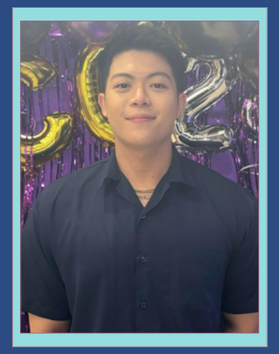
SECP3133-HDPD PROJECT 1 SEC 01

Optimizing Multithreaded Web Scraping (NST News) and Data Processing with Pandas, Dask, and Polars



LEE YIK HONG
A21BE0376

WONG JUN JI
A22EC0117

CHE MARHUMI BIN CHE AB RAHIM
A22EC0147

Lecturer: PROF. MADYA. TS. DR. MOHD SHAHIZAN BIN OTHMAN

INTRODUCTION

New Straits Times (NST) — specifically the Business, World, and ASEAN sections.

Specifically the Business, World, and ASEAN sections.

Selenium combined with multithreading to enable fast and parallel scraping.

Category, Headline, Summary, and Date, storing the results in a structured CSV format.

Processing time, CPU & memory usage, and throughput.

PROJECT OVERVIEW & OBJECTIVES

Problem

❌ Sequential and Resource-Heavy

Most traditional scrapers:

- Load one page at a time (no parallelism)
- Render full pages (including images, CSS, and fonts)
- Fail or timeout with JavaScript-heavy websites (like NST)

Objective

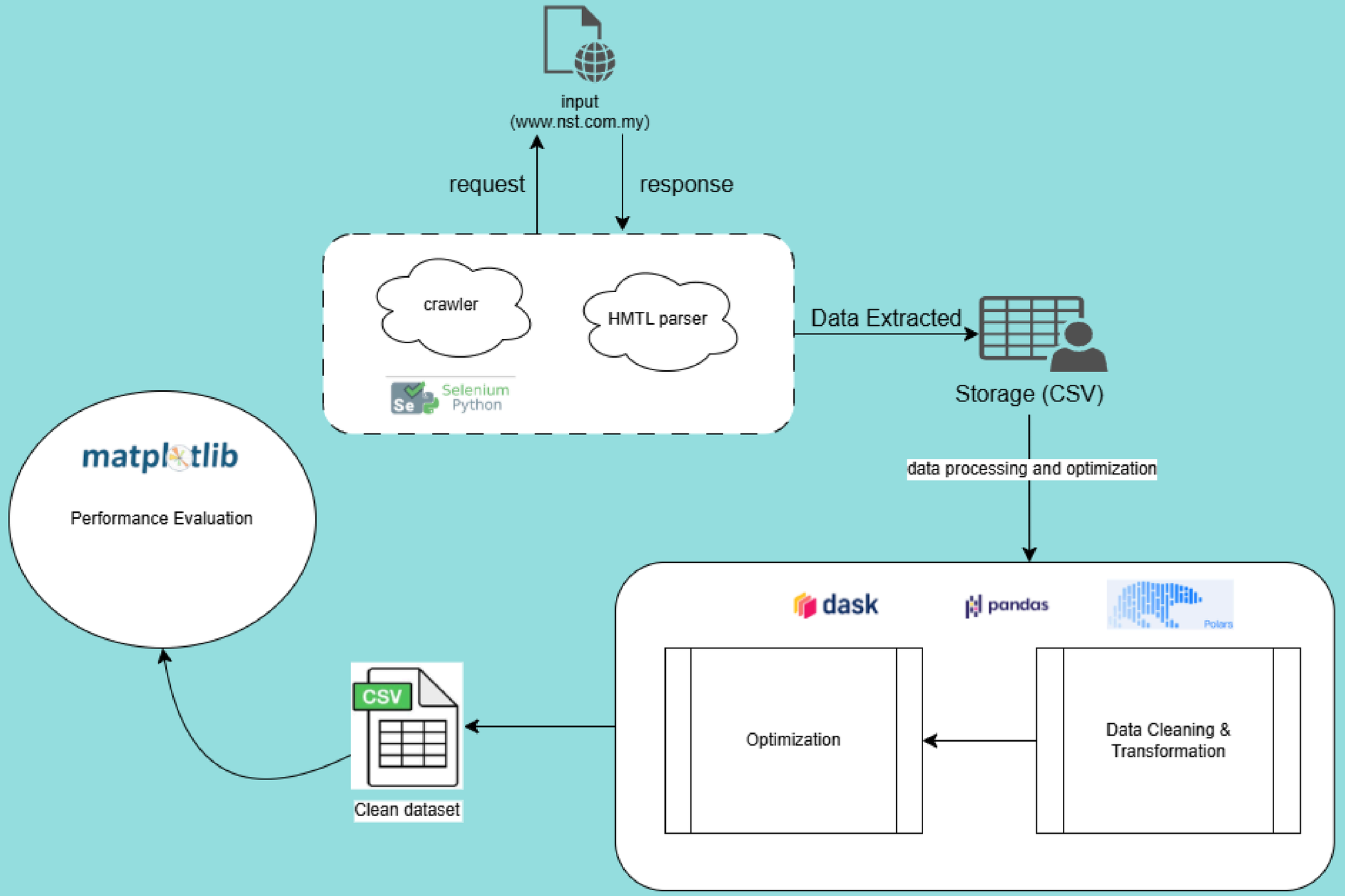
1. Develop efficient, scalable web scraping.
2. Structured data extraction (Category, Headline, Summary, Date).
3. Apply data cleaning using Pandas, Dask, and Polars.
4. Implement performance benchmarking.

Solution

Optimization-Focused Web Scraper (Selenium + Threading + Blocking Resources)

Optimization	Description	Benefit
🧵 Multithreading	Uses 4 threads to scrape pages concurrently	🚀 Speeds up scraping by 2x–5x
🍥 Selective Data Extraction	Only grabs required fields: headline, category, summary, date	🎯 Avoids unnecessary data
😶‍🌫️ Headless Mode	Runs browser without UI	🧠 Saves memory and CPU
🚫 Blocked Resources	Blocks images, fonts, CSS	⚡ Faster page load

System Architecture



Cleaning Methods

Duplicate Removal

• Some articles appear more than once under different categories.
• Duplicates are removed to ensure uniqueness.

Handle Missing/Empty Data

- Empty headlines are removed.
- Missing summaries are filled.
- Ensures consistency and data quality.

Data Structure

Input Format

Raw scraped data loaded from a CSV file.

Output Format

Cleaned and structured data saved as CSV.

Final Data Fields

- **Category**
- **Headline**
- **Summary**
- **Date**
- **Place**
- **Year**
- **Month**

Transformation & Formatting

Date Formatting

Clean and convert date strings to proper datetime format.

Extract Place

Location extracted from the start of summary texts.

Optimisation Technique

Import Required Libraries

```
import pandas as pd
import dask.dataframe as dd
import polars as pl
import time
import psutil
import threading
import matplotlib.pyplot as plt
```

THE PROJECT BEGINS BY IMPORTING ESSENTIAL LIBRARIES FOR DATA PROCESSING, RESOURCE MONITORING, CONCURRENCY AND VISUALIZATION:

```python
def monitor_resources(stop_flag, cpu_list, mem_list):
    while not stop_flag.is_set():
        cpu_list.append(psutil.cpu_percent(interval=0.5))
        mem_list.append(psutil.virtual_memory().percent)
```

Resource Monitoring and Performance Measurement

A MONITORING THREAD COLLECTS CPU AND MEMORY USAGE PERIODICALLY TO CAPTURE SYSTEM RESOURCE USAGE DURING EXECUTION

```python
def monitor_performance(func):
    def wrapper(*args, **kwargs):
        stop_flag = threading.Event()
        cpu_usage = []
        mem_usage = []

        monitor_thread = threading.Thread(target=monitor_resources, args=(stop_flag, cpu_usage, mem_usage))
        monitor_thread.start()

        start_cpu = psutil.cpu_percent(interval=None)  # non-blocking call
        start_mem = psutil.virtual_memory().percent
        start_time = time.time()

        result = func(*args, **kwargs)

        elapsed = time.time() - start_time
        end_cpu = psutil.cpu_percent(interval=None)
        end_mem = psutil.virtual_memory().percent

        stop_flag.set()
        monitor_thread.join()

        avg_cpu = sum(cpu_usage) / len(cpu_usage) if cpu_usage else 0
        peak_mem = max(mem_usage) if mem_usage else 0
        throughput = len(result) / elapsed if elapsed > 0 else 0

        print(f"\n{func.__name__} results:")
        print(f"Time elapsed: {elapsed:.2f} seconds")
        print(f"Avg CPU usage during run: {avg_cpu:.2f}%")
        print(f"Start CPU: {start_cpu:.2f}%, End CPU: {end_cpu:.2f}%")
        print(f"Peak Memory Usage: {peak_mem:.2f}%")
        print(f"Records processed: {len(result):,}")
        print(f"Throughput: {throughput:.2f} records/sec")

        return result, elapsed, avg_cpu, peak_mem, throughput
    return wrapper
```

A DECORATOR NAMED MONITOR_PERFORMANCE WRAPS THE CLEANING FUNCTIONS TO MEASURE THE TOTAL EXECUTION TIME, AVERAGE CPU USAGE, PEAK MEMORY USAGE, AND ALSO THE THROUGHPUT.

Data Cleaning Functions for Each Library

PANDAS CLEANING PIPELINE

```python
@monitor_performance
def clean_pandas(file_path):
    df = pd.read_csv(file_path, encoding='latin1')

    df['headline'] = df['headline'].astype(str).str.strip()
    df = df[df['headline'] != ""]
    df['summary'] = df['summary'].fillna("").astype(str).str.strip()
    df['category'] = df['category'].astype(str).str.strip()

    df = parse_and_clean_dates(df, 'date')

    df['place'] = df['summary'].str.extract(r'^([A-Z\s]+):')[0]
    df['place'] = df['place'].where(df['place'].notnull(), np.nan)
    df['place'] = df['place'].str.strip().str.title()

    df['summary'] = df['summary'].str.replace(r'^[A-Z\s]+:\s*', '', regex=True)
    df['category'] = df['category'].str.title()

    df = df.drop_duplicates()
    df['year'] = df['date'].dt.year
    df['month'] = df['date'].dt.month

    return df
```

DASK CLEANING PIPELINE

```python
@monitor_performance
def clean_pandas(file_path):
    df = pd.read_csv(file_path, encoding='latin1')

    df['headline'] = df['headline'].astype(str).str.strip()
    df = df[df['headline'] != ""]
    df['summary'] = df['summary'].fillna("").astype(str).str.strip()
    df['category'] = df['category'].astype(str).str.strip()

    df = parse_and_clean_dates(df, 'date')

    df['place'] = df['summary'].str.extract(r'^([A-Z\s]+):')[0]
    df['place'] = df['place'].where(df['place'].notnull(), np.nan)
    df['place'] = df['place'].str.strip().str.title()

    df['summary'] = df['summary'].str.replace(r'^[A-Z\s]+:\s*', '', regex=True)
    df['category'] = df['category'].str.title()

    df = df.drop_duplicates()
    df['year'] = df['date'].dt.year
    df['month'] = df['date'].dt.month

    return df
```

```python
@monitor_performance
def clean_polars(file_path):
    df = pl.read_csv(file_path, encoding='latin1')

    df = df.with_columns([
        pl.col('headline').str.strip_chars(),
        pl.col('summary').fill_null("").cast(pl.Utf8).str.strip_chars(),
        pl.col('category').str.strip_chars(),
        pl.col('date').str.strip_chars()
    ])

    df = df.filter(pl.col('headline') != "")

    # Clean date string and rename to 'date_cleaned'
    df = df.with_columns([
        pl.col('date').str.replace('@', '', literal=False).str.strip_chars().alias('date_clean
    ])

    # Parse 'date_cleaned' into datetime, rename to 'date' and drop intermediate column
    df = df.with_columns([
        pl.col('date_cleaned').str.strptime(pl.Datetime, format="%b %d, %Y %I:%M%p", strict=Fa
    ]).drop('date_cleaned')

    place = (
        df.select(pl.col('summary').str.extract(r'^([A-Z\s]+):'))
          .to_series()
          .str.strip_chars()
          .str.to_titlecase()
    )

    # Replace empty strings with null (missing)
    place = place.map_elements(lambda x: None if x == "" else x)

    df = df.with_columns([place.alias('place')])

    df = df.with_columns(
        df['summary'].str.replace(r'^[A-Z\s]+:\s*', '', literal=False).alias('summary')
    )

    df = df.with_columns(
        (
            pl.col('category').str.to_lowercase()
            .str.slice(0, 1).str.to_uppercase()
            + pl.col('category').str.slice(1, None)
        ).alias('category')
    )
```

POLAR CLEANING PIPELINE

```python
def plot_results(results):
    tools = ['Pandas', 'Dask', 'Polars']
    times = [r[1] for r in results]
    cpu = [r[2] for r in results]
    mem = [r[3] for r in results]
    throughput = [r[4] for r in results]

    plt.figure(figsize=(12, 8))

    plt.subplot(2, 2, 1)
    plt.bar(tools, times)
    plt.ylabel('Seconds')
    plt.title('Processing Time')

    plt.subplot(2, 2, 2)
    plt.bar(tools, cpu)
    plt.ylabel('% CPU Usage')
    plt.title('Average CPU Usage')

    plt.subplot(2, 2, 3)
    plt.bar(tools, mem)
    plt.ylabel('% Memory Usage')
    plt.title('Peak Memory Usage')

    plt.subplot(2, 2, 4)
    plt.bar(tools, throughput)
    plt.ylabel('Records/sec')

    plt.tight_layout()
    plt.show()
```

Add Comment

Visualization of Performance Metrics

A PLOTTING FUNCTION THAT DISPLAYS BAR CHARTS COMPARING PROCESSING TIME, AVERAGE CPU USAGE, PEAK MEMORY USAGE, AND THROUGHPUT FOR THE THREE LIBRARIES

```python
if __name__ == "__main__":
    file_path = r"C:\Users\User\Documents\UTM data engineering\s6\HPDP\nst_articles_final.csv"

    print("Starting Pandas cleaning...")
    pandas_df, pandas_time, pandas_cpu, pandas_mem, pandas_throughput = clean_pandas(file_path)
    pandas_df.to_csv("cleaned_pandas.csv", index=False, encoding="utf-8")

    print("Starting Dask cleaning...")
    dask_df, dask_time, dask_cpu, dask_mem, dask_throughput = clean_dask(file_path)
    dask_df.to_csv("cleaned_dask.csv", index=False, encoding="utf-8")

    print("Starting Polars cleaning...")
    polars_df, polars_time, polars_cpu, polars_mem, polars_throughput = clean_polars(file_path)
    polars_df.write_csv("cleaned_polars.csv")

    plot_results([
        (pandas_df, pandas_time, pandas_cpu, pandas_mem, pandas_throughput),
        (dask_df, dask_time, dask_cpu, dask_mem, dask_throughput),
        (polars_df, polars_time, polars_cpu, polars_mem, polars_throughput),
    ])
```

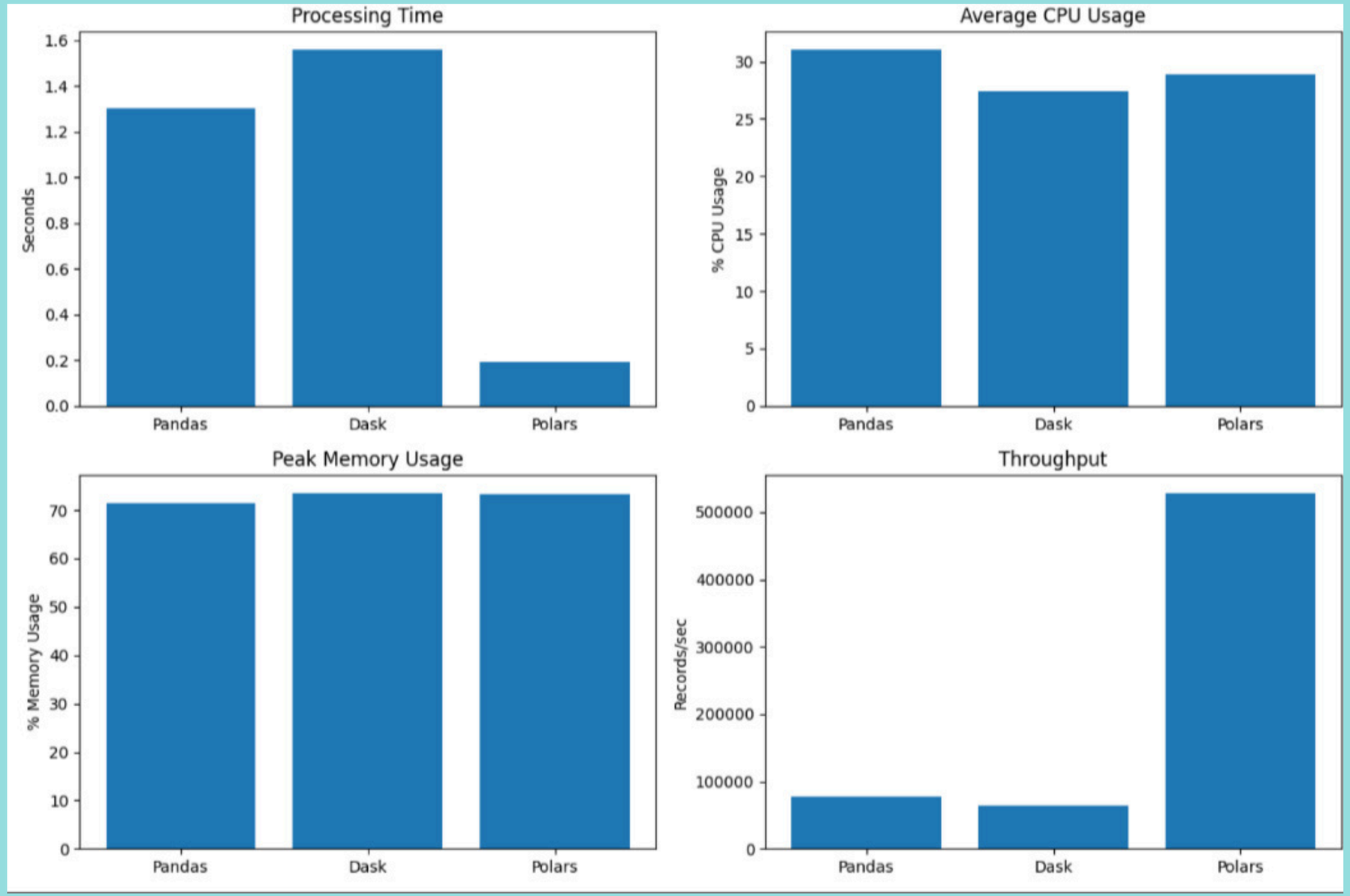
Execution Flow

THE MAIN SCRIPT THAT EXECUTES EACH CLEANING FUNCTION SEQUENTIALLY, SAVES CLEANED DATA FILES, AND VISUALIZES THE PERFORMANCE METRICS

Performance Evaluation

Metric	Pandas	Dask	Polars
Time Elapsed (s)	1.30	1.56	0.19
Average CPU Usage (%)	31.03	27.37	28.90
Peak Memory Usage (%)	71.50	73.60	73.30
Throughput (records/sec)	77550	64766	529123
Records Processed	100962	100962	100962

Visualisation



Challenges & Limitations

WHAT CHALLENGES DID WE FACED DURING THIS PROJECT?

Target Website Selection and Scraping Challenges

The team faced significant challenges in scraping the website due to strong anti-scraping defenses

Tooling and Environment Constraints

Compatibility issues and resource limitations

Data Cleaning And Transformation Challenge

Maintaining consistency and accuracy while performing data cleaning and transformation

Conclusion

• Built a fast multithreaded web scraper using Selenium.
• Collected 100,000+ articles from the NST website.
• Polars processed data fastest (>500,000 records/sec).
• Faced and solved issues like site restrictions and setup problems.

Future Work

- Use distributed scraping (e.g. Playwright).
- Store data in databases instead of CSV.
- Try real-time scraping for faster updates.
- Aim to make the system more scalable and flexible.