Optimizing High-Performance Data Processing for Large-Scale Web Crawlers - News Straits Times (NST Online)

Prepared by:
VINESH A/L VIJAYA KUMAR A22EC0290
JOSEPH LAU YEO KAI A22EC0055
TIEW CHUAN SHEN A22EC0113
NUR FARAH ADIBAH BINTI IDRIS A22EC0245

INTRODUCTION

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

Objectives

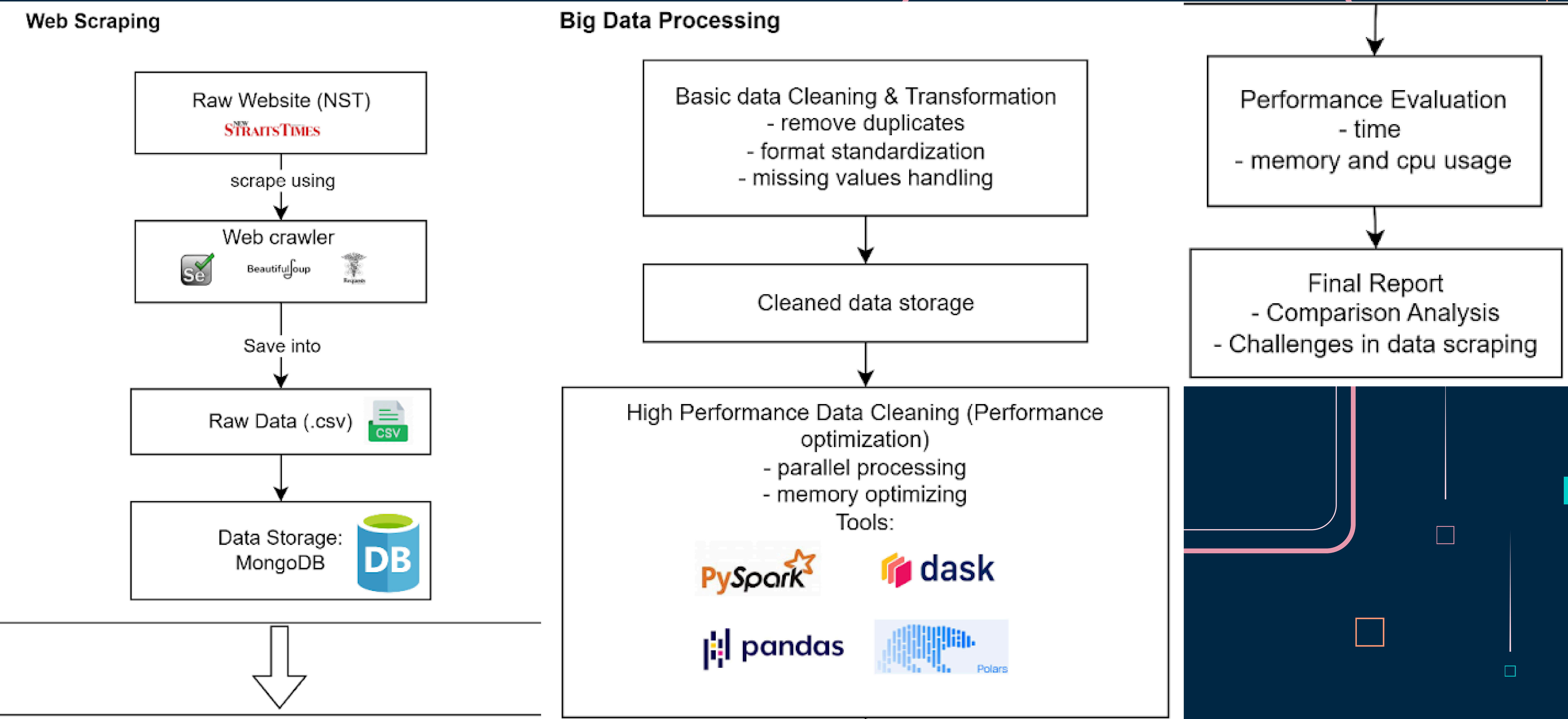
1. To develop a web crawler that is able to extract at least 100,000 records from a News Straits Times (NST) website.

2. To store extracted data in CSV format for further processing.

3. To clean and preprocess the raw dataset.

4. To evaluate performance before and after optimization using several performance metrics.

SYSTEM ARCHITECTURE



DATA COLLECTION

Data Field	Data Type	Description
Section	String	News topic(crime, politics, nation, health).
Publication date	Date	The date(including time) the article is published with format mm:dd:yyyy @ hh:mm
Headline	String	Title of the article.
Summary	String	Brief summary of the news.

- Data aimed to be be extracted from website
- Total rows of data extracted:127729

Crawling Method

1.Pagination Handling

2. Rate Limiting

Ethical Consideration

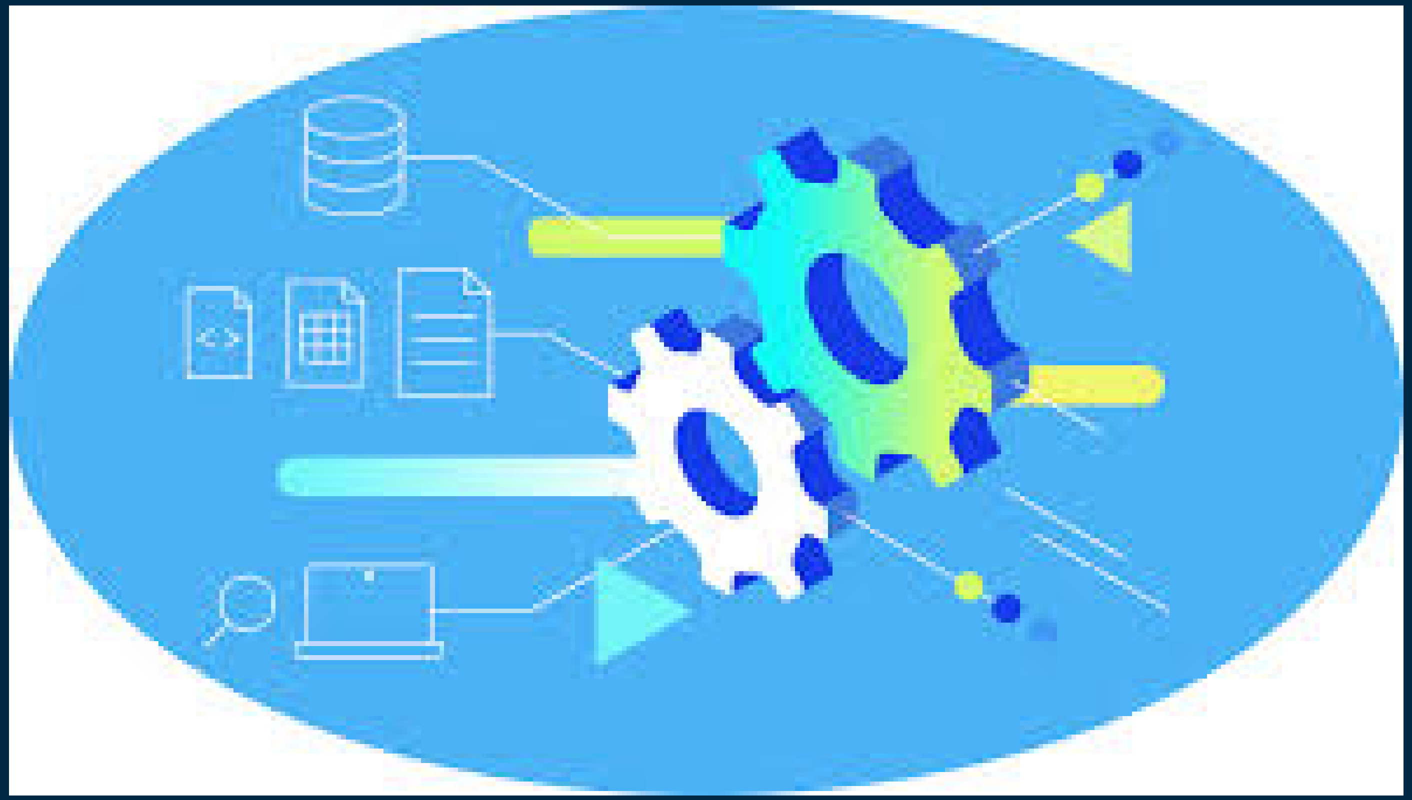
1. Implemented delays between requests
2. Used headless browser configuration
3. Implemented error handling
4. Collected only publicly available information
5. Followed NST's robots.txt guidelines

Data Processing

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

The cleaning and transformation methods

1. Null value
2. Duplicate value
3. Inconsistent capital/small number under the column-section
4. Wrong Date format



OPTIMIZATION TECHNIQUE

Polars
columnar data storage

Dask
processing in parallel and lazily

PySpark
runs on top of the Apache Spark framework(enables distributed computing)

Vectorized Pandas
Pandas functions that operate on entire columns instead of using .apply()

OPTIMIZATION
CODE OVERVIEW

Optimization: Polars

- **.str.split('@')**: Splits the string at the "@" symbol, creating a list.
- **.list.first()**: Extracts the first element of the list (the date part).
- **.str.strip_chars()**: Removes leading/trailing whitespace.
- **.str.strptime(pl.Datetime, format='%b %d, %Y', strict=False)**: Parses the cleaned string into Polars' Datetime type based on the specified format. Polars' string operations are often faster than standard Python loops.

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

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

Optimization: Dask

- **.map_partitions(lambda df: ...)**: Applies the function to each partition of the Dask DataFrame.
- **df['Date'].map(lambda x: x.split('@')[0].strip() if isinstance(x, str) else x)**: split and strip the date string.
- **pd.to_datetime(df['Date'], errors='coerce')**: convert strings to datetime objects.

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

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

Optimization: PySpark

- **regexp_replace(col("Date"), "@.*$", ""):** Uses regular expressions to remove the "@" and everything after it.
- **regexp_replace(col("Date"), "\s+", " "):** Normalizes multiple spaces into a single space.
- **trim(col("Date")):** Removes leading and trailing spaces.
- **to_date(col("Date"), "MMM d,yyyy"):** Converts the cleaned string column to Spark's Date type using the specified format. Spark's operations are designed for large-scale distributed data processing.

```
# Remove time part
df_cleaned = df_cleaned.withColumn("Date",
regexp_replace(col("Date"), "@.*$", ""))

# Normalize spaces
df_cleaned = df_cleaned.withColumn("Date",
regexp_replace(col("Date"), "\s+", " "))

# Trim spaces
df_cleaned = df_cleaned.withColumn("Date",
trim(col("Date")))

# Parse to date
df_cleaned = df_cleaned.withColumn("Date",
to_date(col("Date"), "MMM d, yyyy"))
```

Optimization: Vectorized Pandas

- **df['Date'].str.split('@').str[0].str.strip()**: These are vectorized string operations that apply the split, indexing, and stripping operations to the entire 'Date' Series efficiently without explicit loops.
- **df['Date'].str.replace(r'\s+', ' ', regex=True)**: Another vectorized string replacement using regular expressions for efficient space normalization.
- **pd.to_datetime(df['Date'], errors='coerce')**: converting entire Series of strings to datetime objects.

```
if 'Date' in df.columns:

 df['Date'] =
df['Date'].str.split('@').str[0].str.strip()
 df['Date'] = df['Date'].str.replace(r'\s+',
' ', regex=True)

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

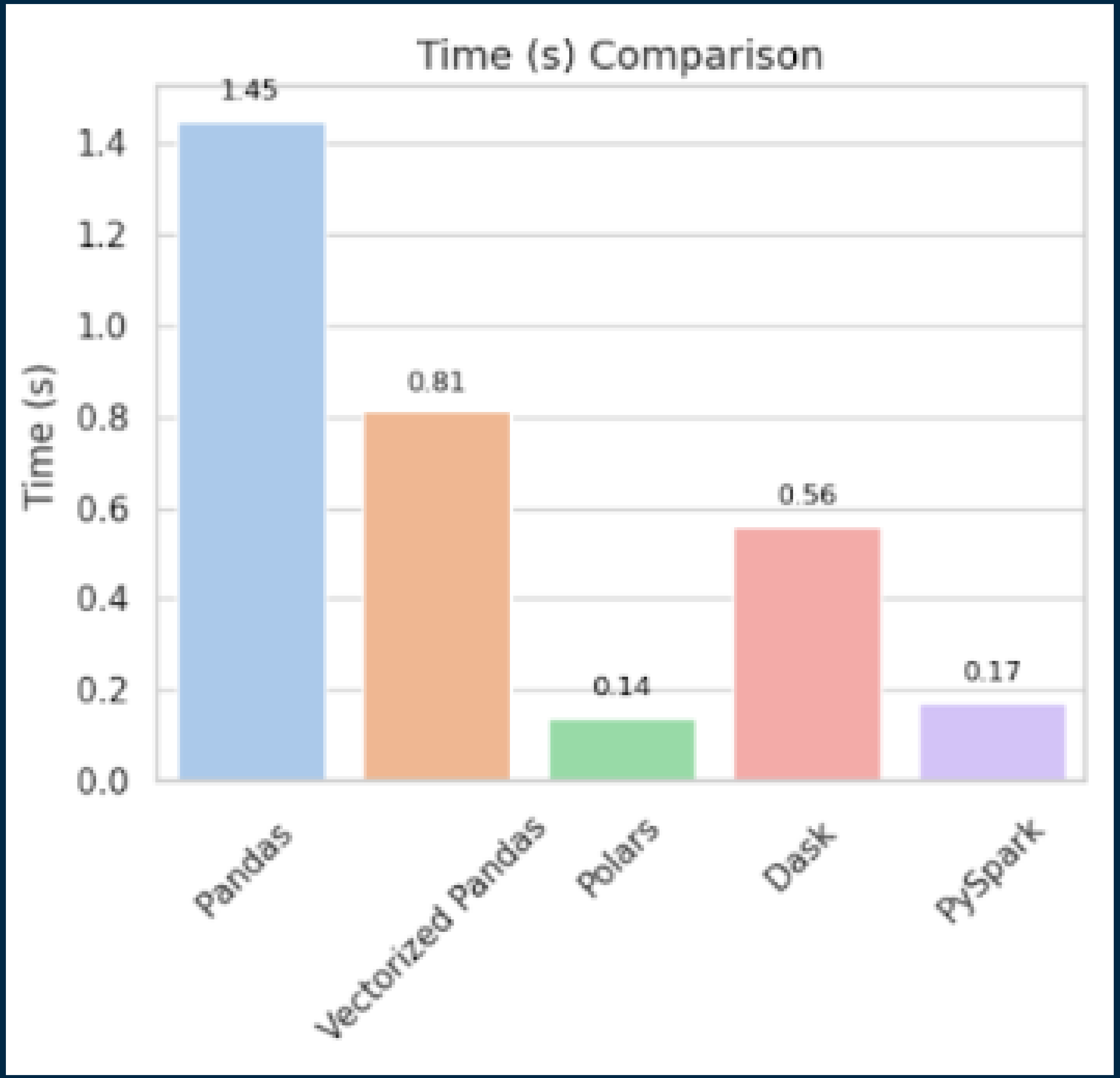
PERFORMANCE EVALUATION

Web Scrapping

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

Table 2: Comparison between Data Processing and Cleaning Techniques

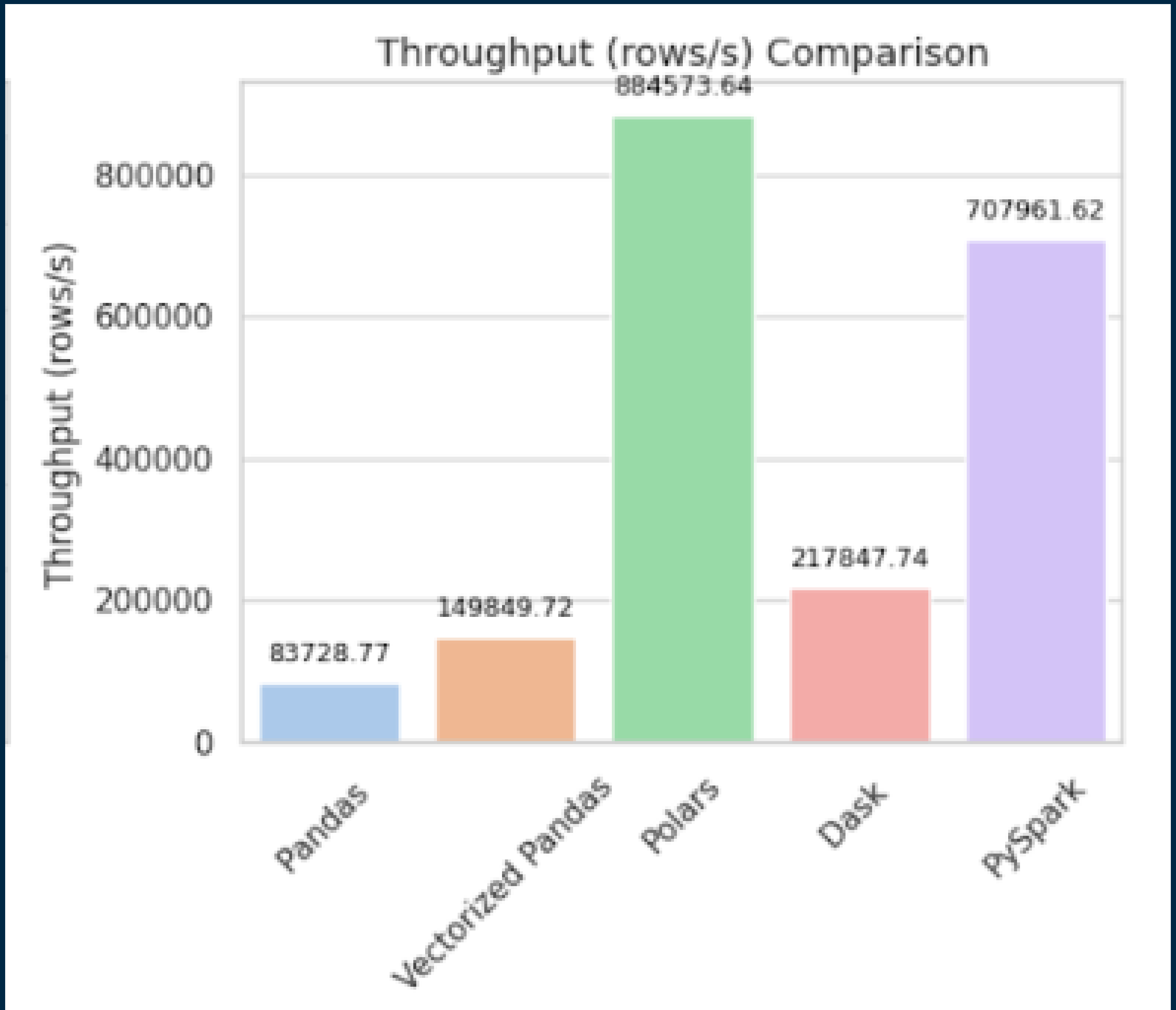
Web Scrapping-Time(s)



Performance Comparison:

- Vectorized Pandas (1.45s)
- Pandas (0.81s)
- Polars (0.14s)
- Dask (0.56s)
- PySpark (0.17s)

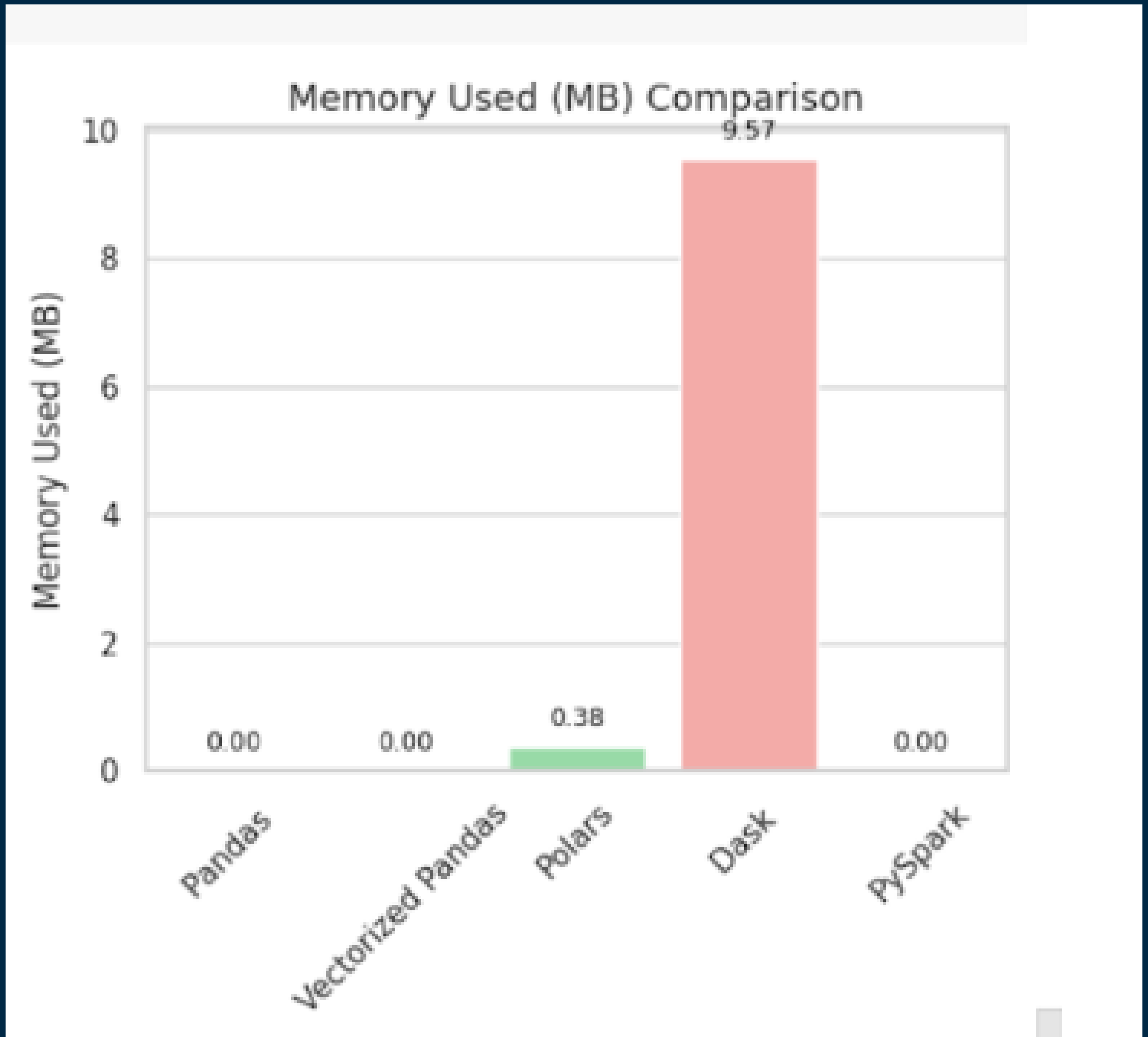
Web Scrapping-Throughput(rows/s)



Throughput Comparison:

- Polars (884573.94 rows/s)
- PySpark (707961.62 rows/s)
- Dask (211847.74 rows/s)
- Vectorized Pandas (140520.72 rows/s)
- Pandas (8528.77 rows/s)

Web Scrapping-Memory(MB) usage



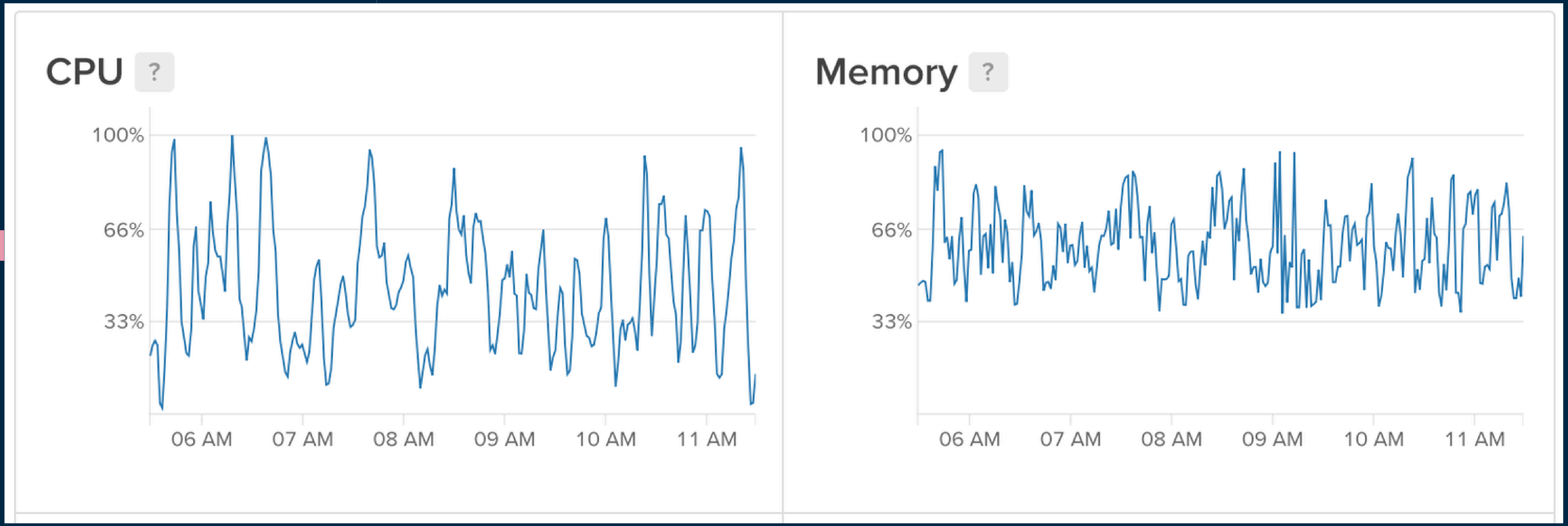
Memory Usage Comparison:

- Dask consumes the most memory (9.57 MB)
- Polars (0.38 MB)
- Pandas, Vectorized Pandas, and PySpark use (0.00 MB)

CHALLENGES & LIMITATIONS

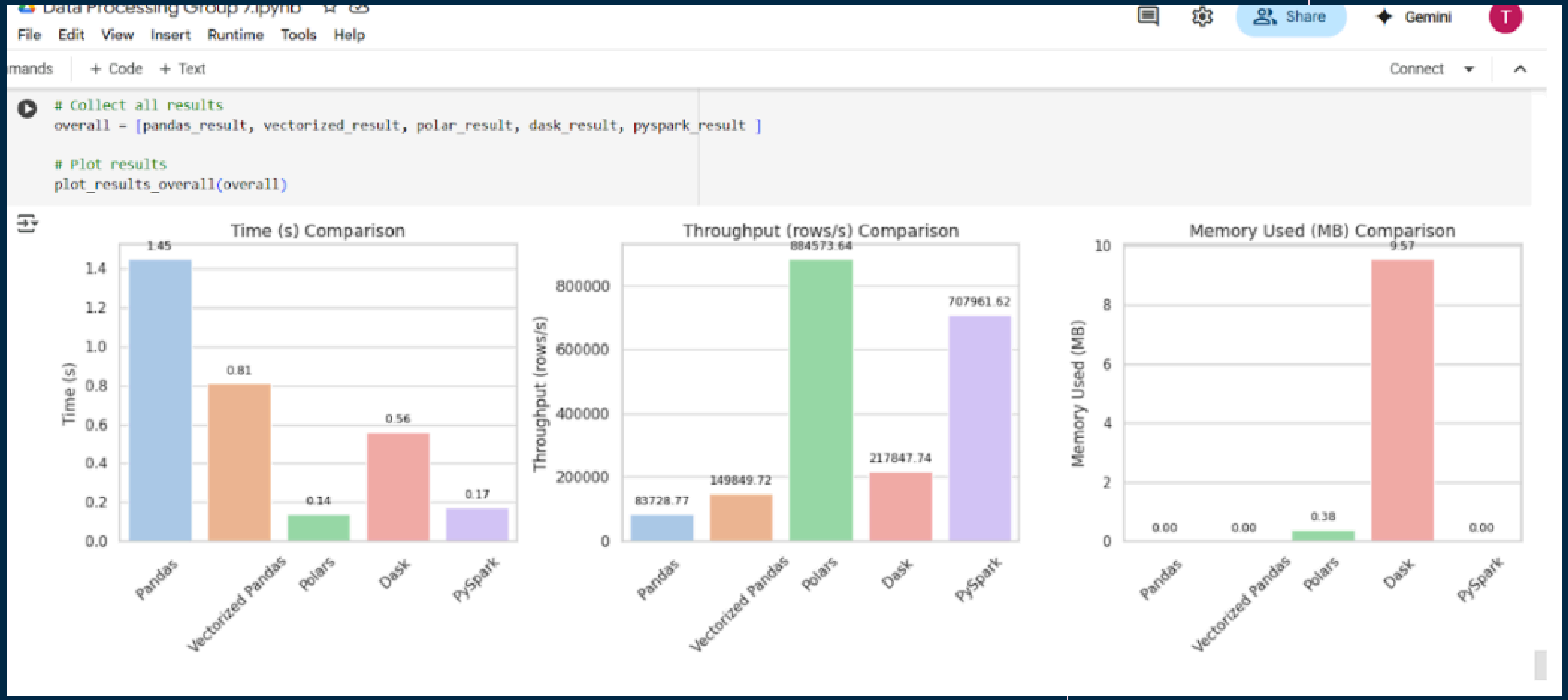
Web Crawler for NST Website – Overview & Challenges

- Developed to scrape NST articles; initially used Scrapy, BeautifulSoup, Selenium.
- Fully shifted to **Selenium** due to dynamic content.
- **Key Challenges**:
  Selenium reliable but **resource-heavy** and **slow**.
- **Ethical scraping**: single-user mode with delays – low scalability.
- **Data cleaning issue**: inconsistent ‘Date’ formats.
- **Pandas** too slow for large datasets (~100k+ rows).



Performance, Bottlenecks & Improvements

- **Optimization Testing**:
  - **Polars**: fastest for large datasets.
  - **Dask**: high memory usage.
  - **PySpark**: slow to start.
  - **Vectorized Pandas**: average speed.
- **Limitations**:
  - Hardcoded to NST’s structure, not reusable.
  - No user interface or hardware-aware logic.
- **Future Improvements**:
  - Flexible scraping engine.
  - Modular, site-agnostic pipelines.
  - GUI for non-technical users



CONCLUSION & FUTURE WORK

Polars Library

01

Time Comparison
- 0.14 seconds

Throughput
- 884573.64 rows per second

Memory Usage
- 0.38 MB

If compared to pandas, Polars can achieve more than 30x performance gains.

Future Work

02

- More complex data transformation
- multiple runs to account for variability and more reliable averages
- larger dataset to stress-test the methods
- investigation into why Dask's memory usage was higher