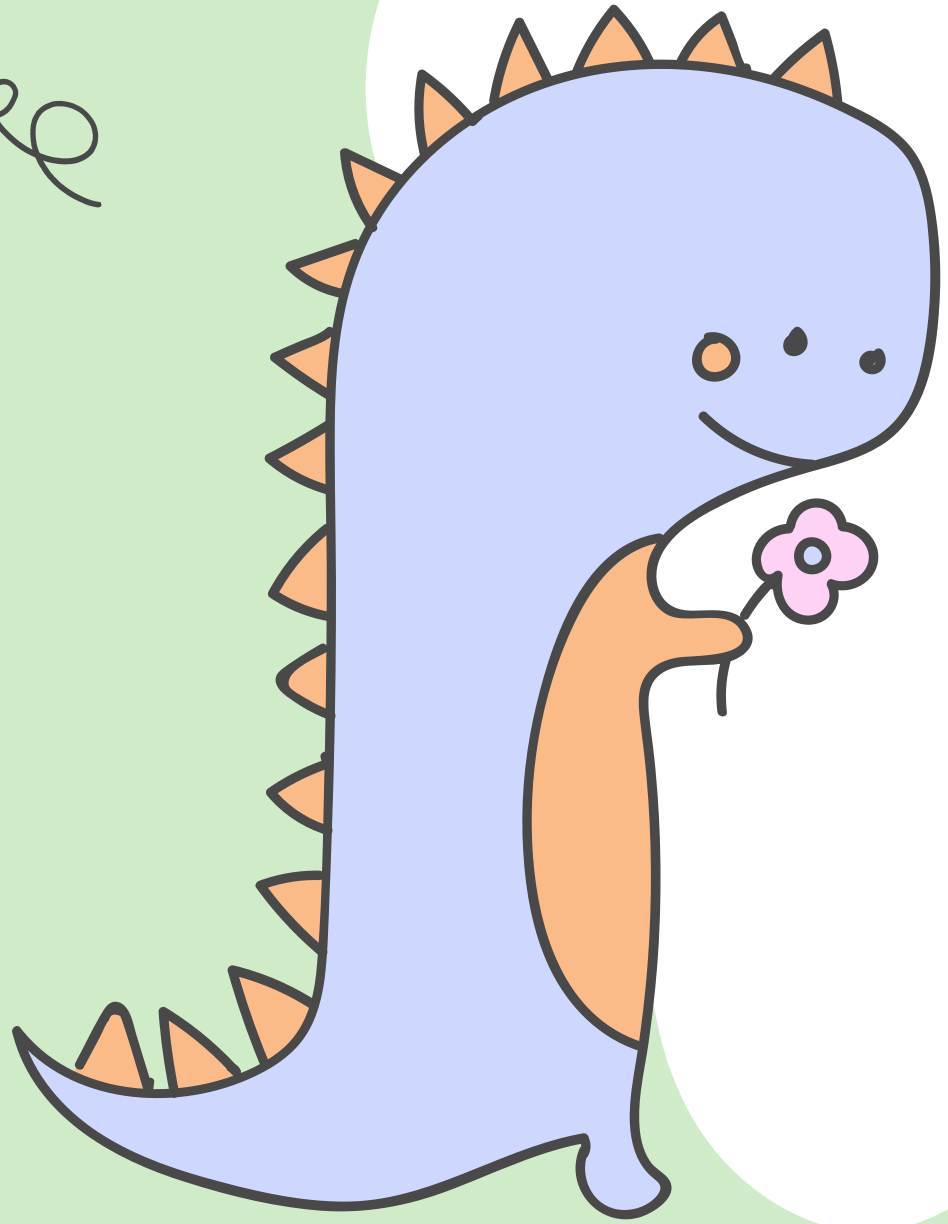
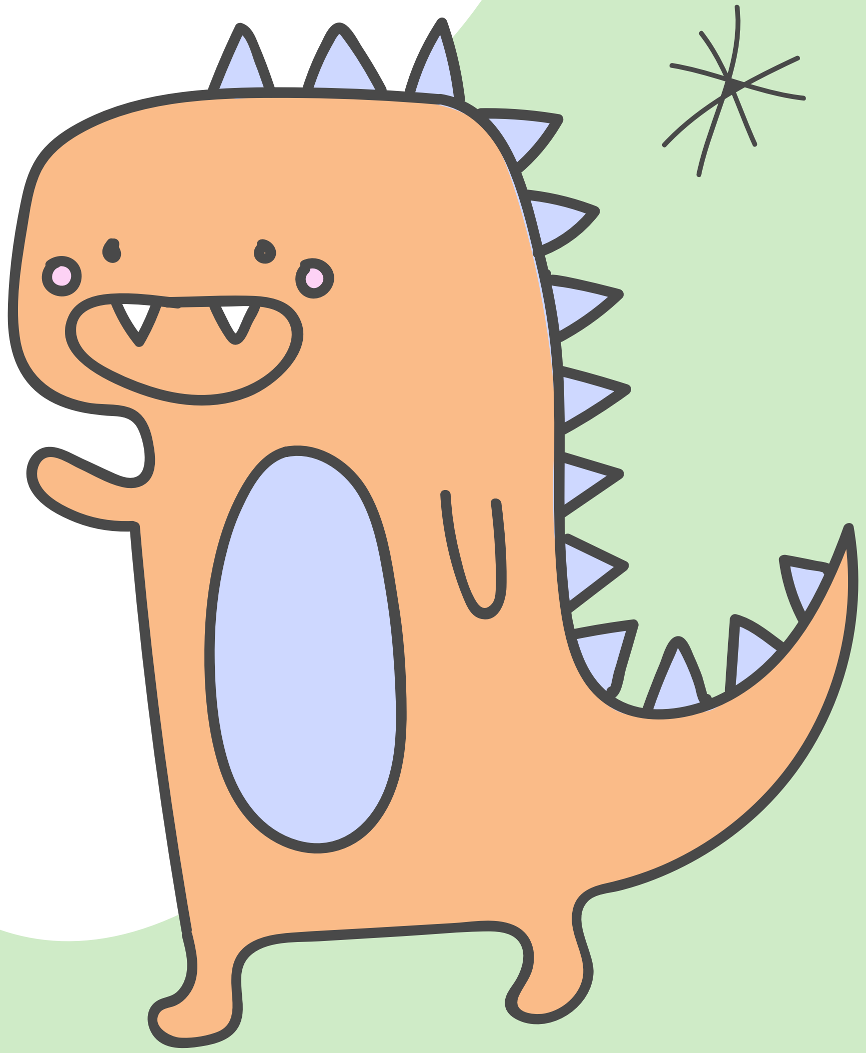
# Optimizing HPDP for Large-Scale Web Crawlers



SECP3133 HPDP
Section 02 Group 4



# Content

1. Target Web
2. System Design and Architecture
3. Crawling Method
4. Ethical Consideration

5. Data Cleaning and Transformation
6. Optimization Method
7. Performance Evaluation
8. Challenges and Limitation

Lazada

Market leader with broad product categories (electronics, fashion, home appliances, beauty)

Rich product data: IDs, prices, discounts, seller info, stock, shipping, ratings

Reliable data from official & verified sellers plus active user reviews

User controls help target crawling, reduce server load, support ethical scraping

Offers structured data & real-world challenges for testing crawler performance

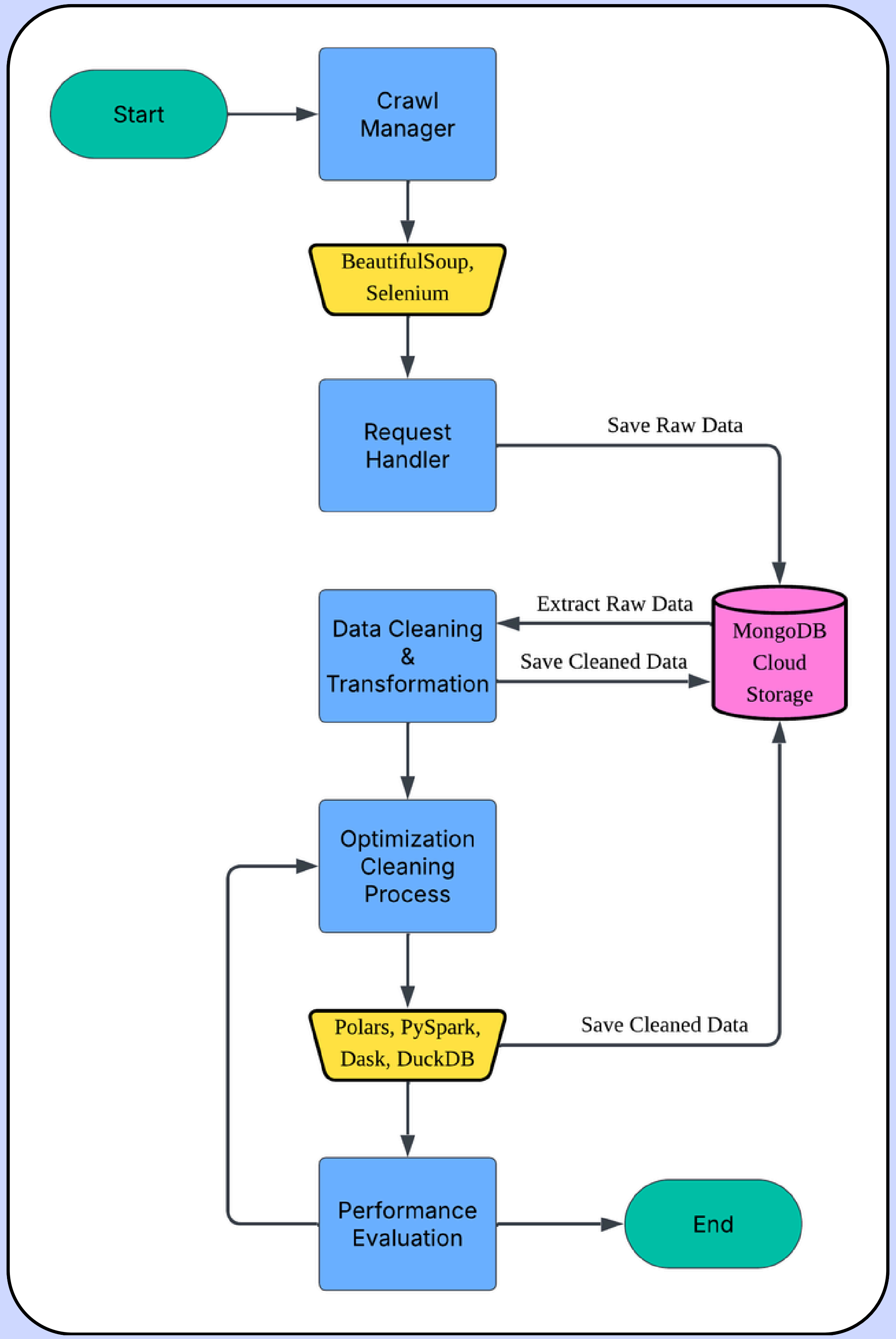
# System Design and Architecture

### 1. Crawl Manager

- Orchestrates the crawling strategy, manages URL queues, monitors request frequency, and invokes crawling engines
- Ensures respect for rate limits and robots.txt

### 3. Request Handler

- Handles complex headers and user-agent rotation for Lazada.
- Manages cookies and session tokens (important for logged-in-only data)
- Captures anti-bot flags and implements fallback (retry or proxy)




### 2. Crawling Libraries

- Selenium: used for rendering dynamic JavaScript-based content
- BeautifulSoup: parses the static HTML/XML content to extract relevant fields

### 4. Data Cleaning & Transformation Plan

- Prepare raw data for analysis through basic preprocessing and feature formatting using Pandas
- Key Operations:
    i. Duplicate Removal
    ii. Missing Value Handling
    iii. Data Formatting
    iv. Type Conversion

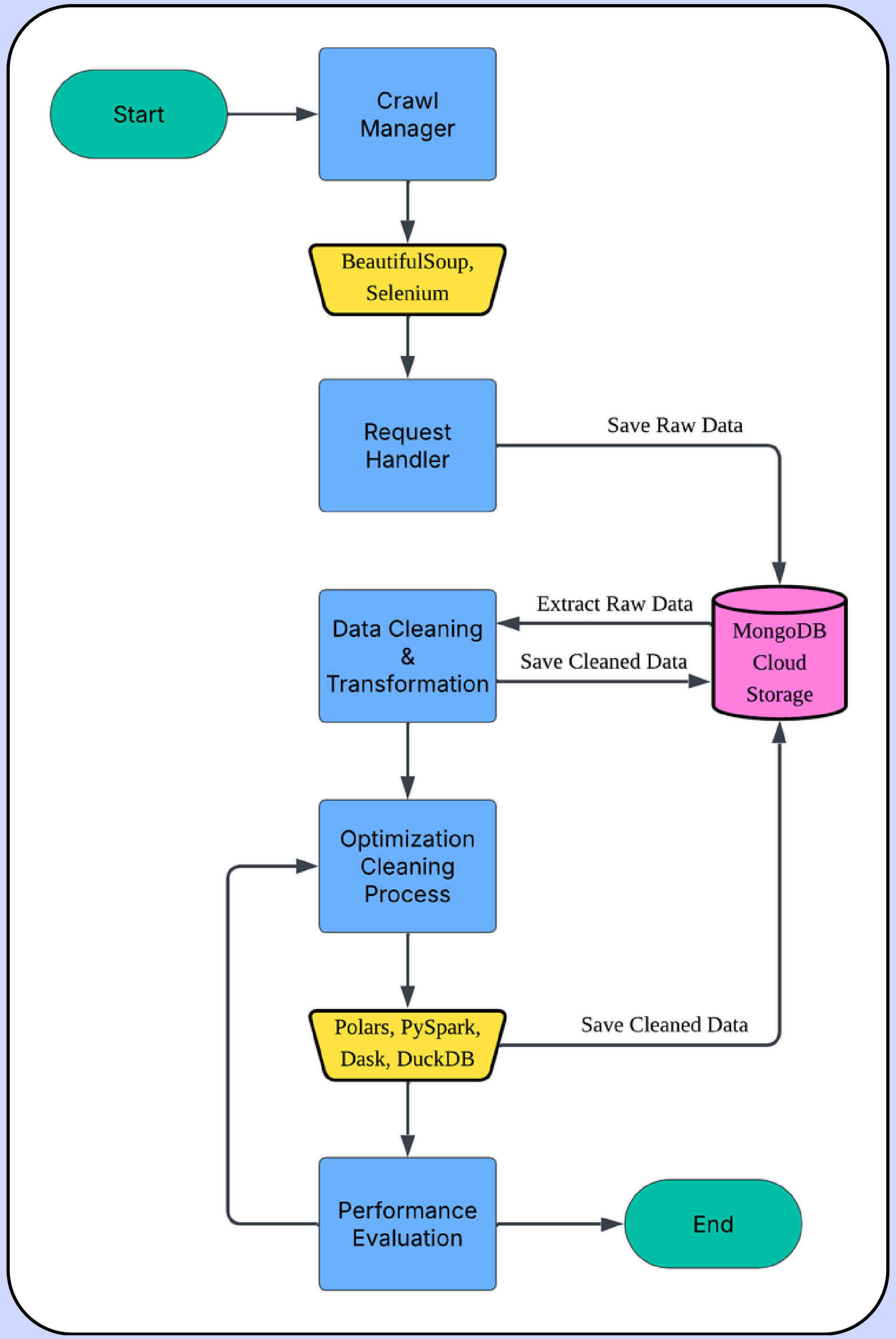
# System Design and Architecture

### 6. Optimization Cleaning Process

Leverages high-performance data processing frameworks to scale and accelerate transformation tasks on large datasets.

Key Operations:

- Parallelized Data Processing
- In-Memory Computation
- Query Optimization




### 5. Data Storage -MongoDB

- Persistently stores raw and cleaned datasets
- Uses MongoDB Atlas with pymango for CRUD operations
- Supports insertion of Pandas DataFrame dictionaries and flexible schema

### 7. Performance Evaluation

- Quantifies efficiency improvements from the optimized data cleaning processes
- Metric captured (time taken for key operations, CPU and memory footprint)
- Data visualization via matplotlib and seaborn

# Crawling Method

# Crawling Method

## Library : Selenium + BeautifulSoup

1. Pagination Handling

a) Get Total Number of Pages

```
pagination = soup.select(".ant-pagination-item")
total_pages = int(pagination[-1].text) if pagination else 1
```

b) Loop Through Each Page

```
for page in range(total_pages):
    print(f"Scraping page {page+1} of {total_pages}")
```

c) Loop Through Each Page

```
next_button = driver.find_element(By.CSS_SELECTOR, ".ant-pagination-next > button")
time.sleep(random.uniform(3, 5))  # Simulate reading delay
next_button.click()
```

# Crawling Method

## Library : Selenium + BeautifulSoup

### 2 Rate Limiting and Anti-Bot Measures

a) Random Delays (Rate Limiting)

```
time.sleep(random.uniform(2.5, 4.5))
```

```
time.sleep(random.uniform(3, 5))
```

b) Set Fake User-Agent

```
options = webdriver.ChromeOptions()
options.add_argument("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"
                    + "(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36")
driver = webdriver.Chrome(options=options)
```

c) CAPTCHA Detection

```
try:
    WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.ID, "captcha"))  # Adjust if Lazada uses different selector
    )
    input("⚠️ CAPTCHA detected. Please solve it manually in the browser, then press Enter to continue...")
except:
    pass
```

# Records Collected

## Total : 120256 rows

Data Recorded :

| 1 | Product Name | Price | Location | Quantity Sold | Number of Ratings |
|---|---|---|---|---|---|
| 2 | [NOT FOR SALE] Korean Fashion Cloth | 0.1 | Penang | 5 sold | N/A |
| 3 | ZD [stock] Letter Printed Short-sleeved T-shirt Men and Women Person | 0.9 | China | N/A | N/A |
| 4 | ZD Summer Yoga Beach Shorts Sports Shorts for Women Home Casual S | 1 | China | N/A | N/A |
| 5 | HD Summer Yoga Beach Shorts Sports Shorts for Women Home Casual | 1 | China | N/A | N/A |
| 6 | 4A Shop Running Shorts for Women Spring Summer Fashion Casual Sho | 1 | China | N/A | N/A |
| 7 | HD Breathable Sports Shorts Women's Summer Home Casual Shorts So | 1 | China | N/A | N/A |
| 8 | ZD Breathable Sports Shorts Women's Summer Home Casual Shorts So | 1 | China | N/A | N/A |
| 9 | HD Sports Shorts Women's Summer 2024 Casual Outerwear Three Pant | 1 | China | N/A | N/A |
| 10 | HD Running Shorts for Women Spring Summer Fashion Casual Shorts B | 1 | China | N/A | N/A |

# Ethical Consideration

**Responsible Request Timing**
Random delays (2.5–4.5s) mimic human browsing to prevent server overload

**CAPTCHA Detection**
Script pauses when CAPTCHA appears

**Public Data Only**
Collected only publicly accessible info (product names, prices, locations, sales, ratings)

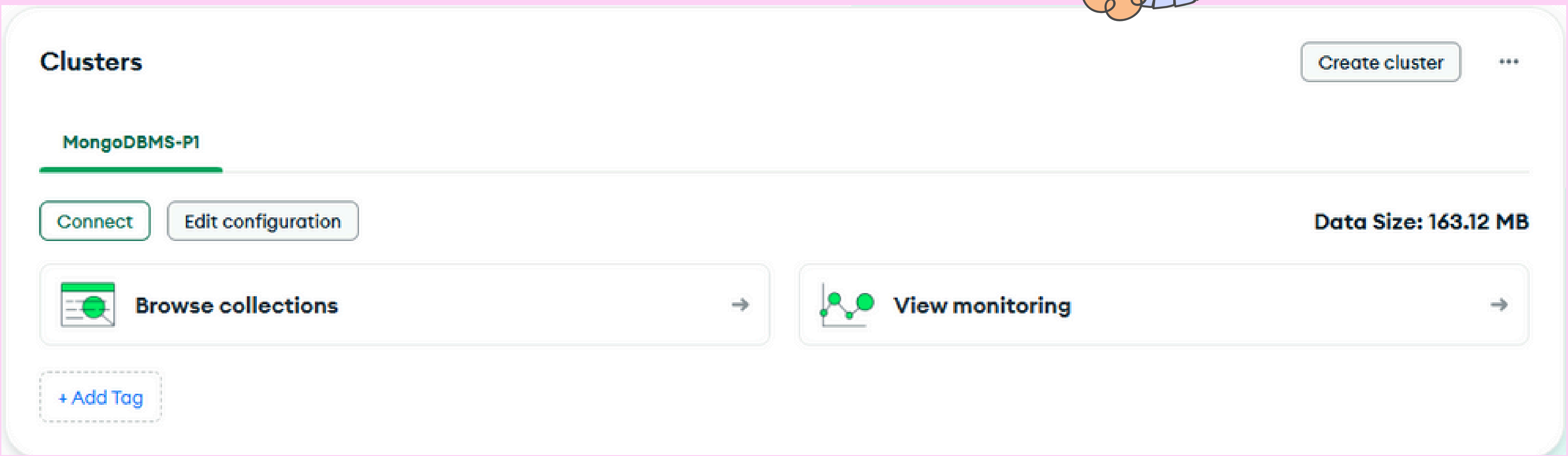
**Ethical Data Usage**
Data used strictly for academic analysis

**Alignment with Best Practices**
Followed ethical scraping guidelines

# Data Cleaning and Transformation

# Data Cleaning and Transformation

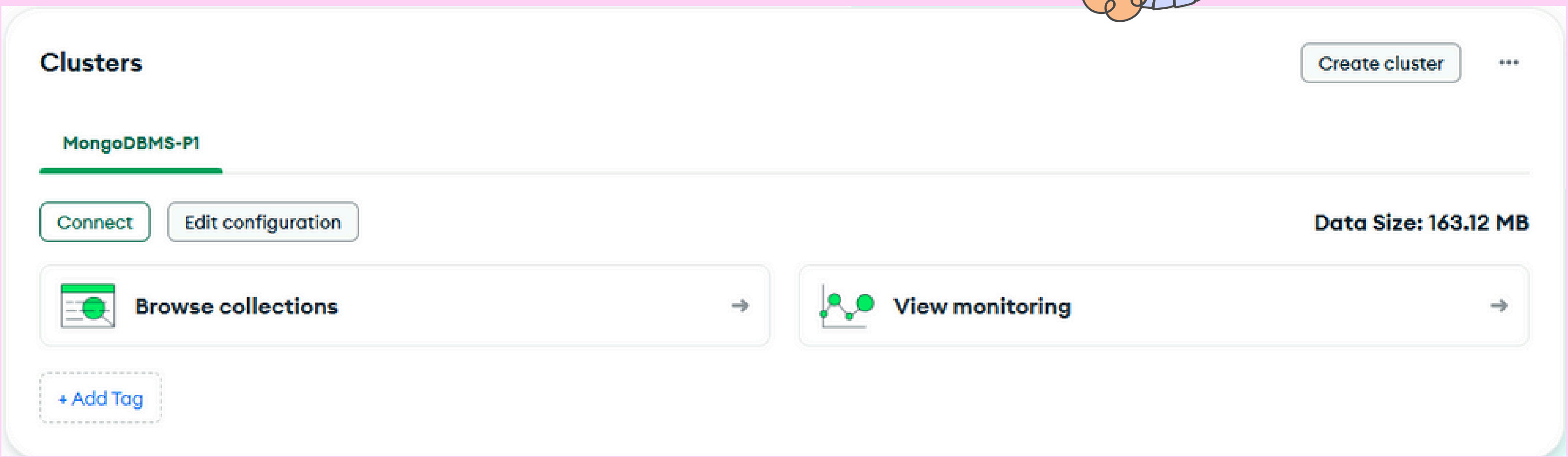
Create Connection




```python
from pymongo import MongoClient

uri = "mongodb+srv://hanwei:hanwei123@mongodbms-p1.5d52qxu.mongodb.net/?retryWrites=true&w=majority&appName=MongoDBMS-P1"
client = MongoClient(uri)

db = client["MongoDBMS-P1"]
collection = db["mycollection"]
```

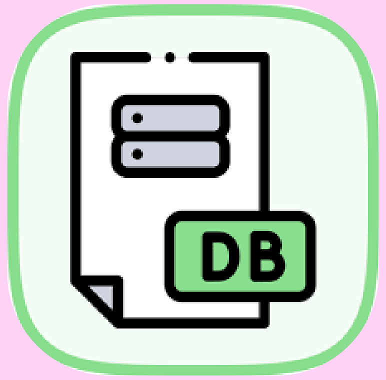
# Data Cleaning and Transformation


import .csv file

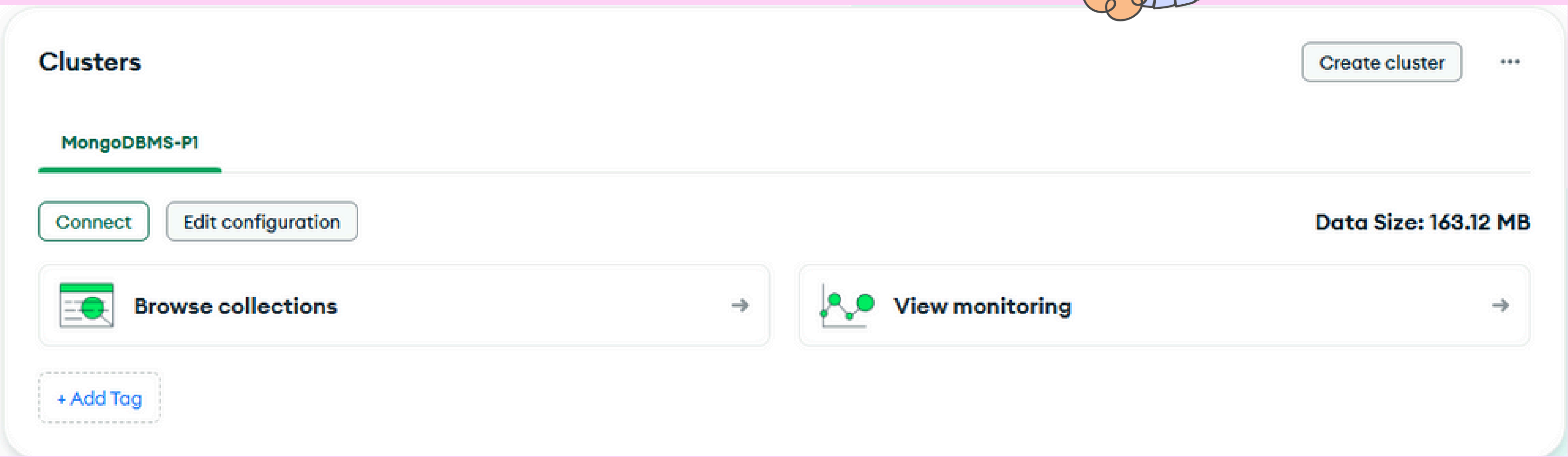



```python
# Load CSV or JSON
df = pd.read_csv("Dataset.csv", encoding="ISO-8859-1")

# Convert to dictionary format for MongoDB
data_dict = df.to_dict("records")

# Insert into MongoDB
collection.insert_many(data_dict)
```

# Data Cleaning and Transformation




load data into DataFrame




```
# Load data from MongoDB into DataFrame
df = pd.DataFrame(list(collection.find()))

# Drop MongoDB's autogenerated _id (optional, re-created on insert)
if '_id' in df.columns:
    df.drop(columns=['_id'], inplace=True)
```

# Data Cleaning and Transformation

1 Drop Duplicates

```python
# 1. Drop duplicates
df.drop_duplicates(inplace=True)
```

| | Product Name | Price | Location | Quantity Sold | Number of Ratings |
|---|---|---|---|---|---|
| 0 | [NOT FOR SALE] Korean Fashion Cloth | 0.1 | Penang | 5 sold | NaN |
| 1 | ZD [stock] Letter Printed Short-sleeved T-shir... | 0.9 | China | NaN | NaN |
| 2 | ZD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | NaN | NaN |
| 3 | HD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | NaN | NaN |
| 4 | 4A Shop Running Shorts for Women Spring Summer... | 1 | China | NaN | NaN |
| ... | ... | ... | ... | ... | ... |
| 120251 | ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O... | RM9,531.00 | NaN | NaN | NaN |
| 120252 | ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL... | RM11,913.00 | NaN | NaN | NaN |
| 120253 | Acer Predator Triton Neo 16 PTN16-51-91BP (Int... | RM8,999.00 | NaN | NaN | NaN |
| 120254 | ASUS Zenbook Duo Ux8406M-Apz042Ws Grey | RM11,494.00 | NaN | NaN | NaN |
| 120255 | Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L... | RM7,699.00 | NaN | NaN | NaN |

116308 rows × 5 columns

# Data Cleaning and Transformation

2 Fill NaN

```python
# 2. Replace NaN in specific columns with "unknown"
df_copy["Product Name"].fillna("unknown", inplace=True)
df_copy["Location"].fillna("unknown", inplace=True)
df_copy["Price"].fillna("unknown", inplace=True)

# 3.  Fill actual NaN with "0" in critical columns
df_copy["Quantity Sold"].fillna("0", inplace=True)
df_copy["Number of Ratings"].fillna("0", inplace=True)
```

| | Product Name | Price | Location | Quantity Sold | Number of Ratings |
|---|---|---|---|---|---|
| 0 | [NOT FOR SALE] Korean Fashion Cloth | 0.1 | Penang | 5 sold | 0 |
| 1 | ZD [stock] Letter Printed Short-sleeved T-shir... | 0.9 | China | 0 | 0 |
| 2 | ZD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 3 | HD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 4 | 4A Shop Running Shorts for Women Spring Summer... | 1 | China | 0 | 0 |
| ... | ... | ... | ... | ... | ... |
| 120251 | ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O... | RM9,531.00 | unknown | 0 | 0 |
| 120252 | ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL... | RM11,913.00 | unknown | 0 | 0 |
| 120253 | Acer Predator Triton Neo 16 PTN16-51-91BP (Int... | RM8,999.00 | unknown | 0 | 0 |
| 120254 | ASUS Zenbook Duo Ux8406M-Apz042Ws Grey | RM11,494.00 | unknown | 0 | 0 |
| 120255 | Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L... | RM7,699.00 | unknown | 0 | 0 |

116308 rows × 5 columns

# Data Cleaning and Transformation

3

Transform
“Quantity Sold”

```
# 4. Clean "Quantity Sold" (e.g., "5K sold" → 5000)
import re
def clean_quantity(q):
    if isinstance(q, str):
        q = q.lower().replace("sold", "").strip()
        if "k" in q:
            return int(float(q.replace("k", "")) * 1000)
        return int(re.findall(r"\d+", q)[0]) if re.findall(r"\d+", q) else 0
    return 0

df_copy["Quantity Sold"] = df_copy["Quantity Sold"].apply(clean_quantity)
```

| | Product Name | Price | Location | Quantity Sold | Number of Ratings |
|---|---|---|---|---|---|
| 0 | [NOT FOR SALE] Korean Fashion Cloth | 0.1 | Penang | 5 | 0 |
| 1 | ZD [stock] Letter Printed Short-sleeved T-shir... | 0.9 | China | 0 | 0 |
| 2 | ZD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 3 | HD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 4 | 4A Shop Running Shorts for Women Spring Summer... | 1 | China | 0 | 0 |
| ... | ... | ... | ... | ... | ... |
| 120251 | ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O... | RM9,531.00 | unknown | 0 | 0 |
| 120252 | ASUS ROG Zephyrus G16 GA605W VQR037W- 16â□□ OL... | RM11,913.00 | unknown | 0 | 0 |
| 120253 | Acer Predator Triton Neo 16 PTN16-51-91BP (Int... | RM8,999.00 | unknown | 0 | 0 |
| 120254 | ASUS Zenbook Duo Ux8406M-Apz042Ws Grey | RM11,494.00 | unknown | 0 | 0 |
| 120255 | Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L... | RM7,699.00 | unknown | 0 | 0 |

116308 rows × 5 columns

# Data Cleaning and Transformation

```
# 5. Clean "Number of Ratings" (e.g., "(10)" → 10)
def clean_ratings(r):
    if isinstance(r, str):
        match = re.search(r"\d+", r)
        return int(match.group()) if match else 0
    return 0

df_copy["Number of Ratings"] = df_copy["Number of Ratings"].apply(clean_ratings)
```

4 Transform "Number of Ratings"

| | | |
|---|---|---|
| (10) | -> | 10 |
| (99) | -> | 99 |
| (1000) | -> | 1000 |
| (1) | -> | 1 |
| (0) | -> | 0 |

# Data Cleaning and Transformation

```python
# 6. Remove non-numerical character in "Price"
# Ensure string type
df_copy["Price"] = df_copy["Price"].astype(str)

# Clean only rows that are not "unknown"
df_copy.loc[df_copy["Price"] != "unknown", "Price"] = (
    df_copy.loc[df_copy["Price"] != "unknown", "Price"]
    .str.replace(r"[^\d.]", "", regex=True)
)
```

| | Product Name | Price | Location | Quantity Sold | Number of Ratings |
|---|---|---|---|---|---|
| 0 | [NOT FOR SALE] Korean Fashion Cloth | 0.1 | Penang | 5 | 0 |
| 1 | ZD [stock] Letter Printed Short-sleeved T-shir... | 0.9 | China | 0 | 0 |
| 2 | ZD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 3 | HD Summer Yoga Beach Shorts Sports Shorts for ... | 1 | China | 0 | 0 |
| 4 | 4A Shop Running Shorts for Women Spring Summer... | 1 | China | 0 | 0 |
| ... | ... | ... | ... | ... | ... |
| 120251 | ASUS Vivobook Pro N6506M VMA030WSM- 15.6" 3K O... | 9531.00 | unknown | 0 | 0 |
| 120252 | ASUS ROG Zephyrus G16 GA605W VQR037W- 16â OL... | 11913.00 | unknown | 0 | 0 |
| 120253 | Acer Predator Triton Neo 16 PTN16-51-91BP (Int... | 8999.00 | unknown | 0 | 0 |
| 120254 | ASUS Zenbook Duo Ux8406M-Apz042Ws Grey | 11494.00 | unknown | 0 | 0 |
| 120255 | Asus Zenbook 14 OLED UX3405M-APZ345 / 346WSM L... | 7699.00 | unknown | 0 | 0 |

116308 rows × 5 columns

5 Transform "Price"

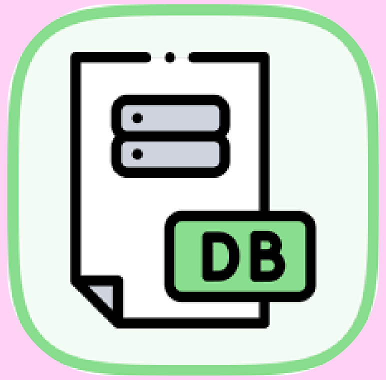
# Data Cleaning and Transformation

```python
# 7. Convert "Price" to float and leave the word "unknown"
def to_float_or_unknown(val):
    try:
        return float(val)
    except:
        return "unknown"

df_copy["Price"] = df_copy["Price"].apply(to_float_or_unknown)


# 8. Convert Quantity Sold & Number of Ratings to int
df_copy["Quantity Sold"] = df_copy["Quantity Sold"].astype(int)
df_copy["Number of Ratings"] = df_copy["Number of Ratings"].astype(int)
```

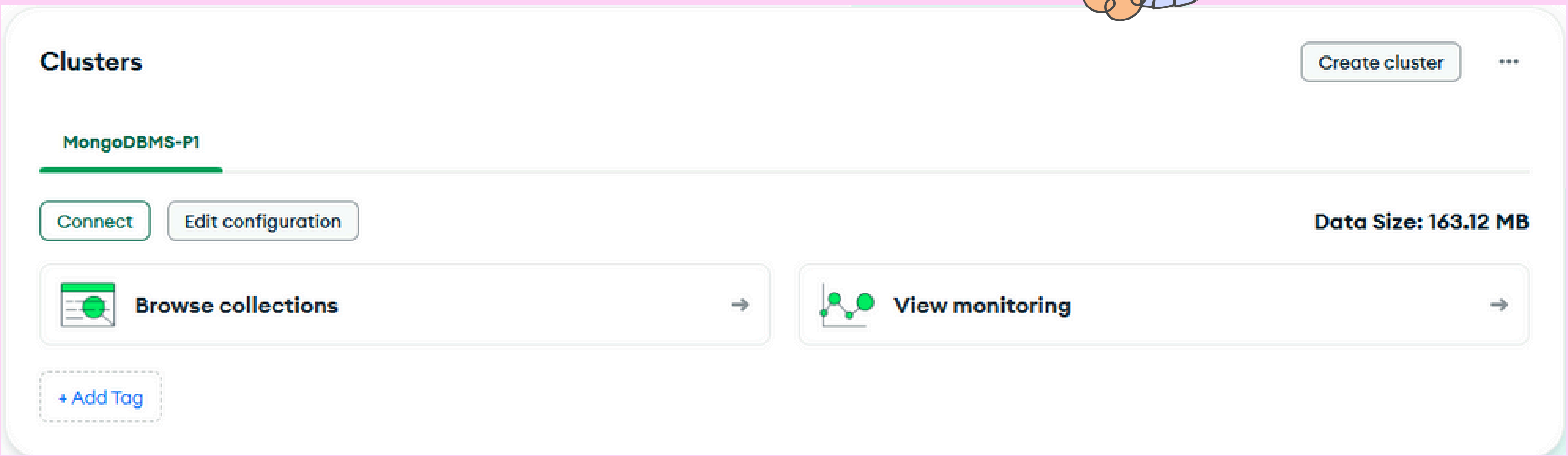
6 DataType Conversion

| | | |
|---|---|---|
| Price (string) | -> | float |
| Quantity Sold (string) | -> | int |
| Number of Ratings (string) | -> | int |

# Data Cleaning and Transformation



load cleaned data back to MongoDb




```
# 8. Upload cleaned data back to MongoDB
collection.drop()
collection.insert_many(df_copy.to_dict("records"))
```

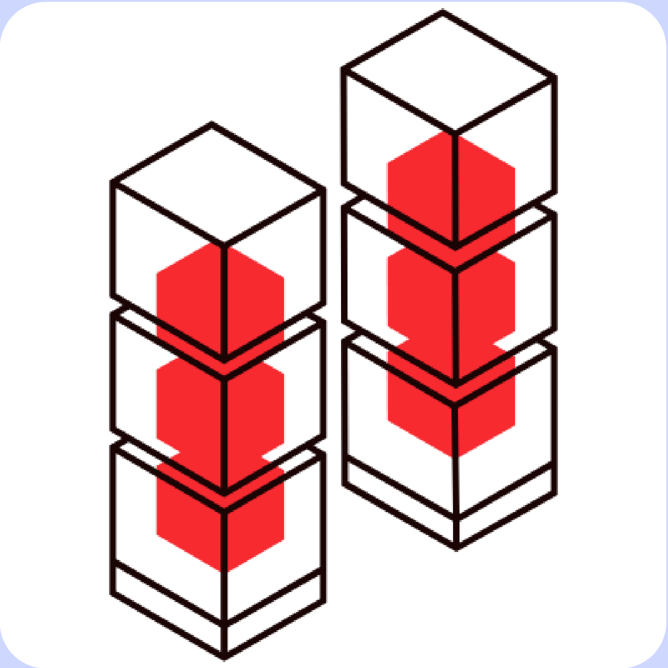
# Optimization Techniques

# Polars

## Why choose Polars?

### Lazy Execution


### Efficient Columnar Memory Layout



### Native Multithreaded Parallelism

# Polars

## 1 Load data in lazy mode

```python
# 2. Load data using pandas with proper encoding, then convert to Polars
print("Loading data with pandas and converting to Polars...")
try:
    pandas_df = pd.read_csv("Dataset.csv", encoding="ISO-8859-1")
    df = pl.from_pandas(pandas_df)
    df_lazy = df.lazy()

    print("\nSchema of the DataFrame:")
    schema = df_lazy.collect_schema()
    for name, dtype in schema.items():
        print(f"- {name}: {dtype}")

    print("\nFirst few rows of raw data:")
    print(df_lazy.fetch(5))

except Exception as e:
    print(f"Error loading data: {e}")
    raise
```

# Polars

## Difference in Pandas and Polars

### Pandas

```
# 2. Replace NaN in specific columns with "unknown"
df_copy["Product Name"].fillna("unknown", inplace=True)
df_copy["Location"].fillna("unknown", inplace=True)
df_copy["Price"].fillna("unknown", inplace=True)
```

```
# 5. Clean "Number of Ratings" (e.g., "(10)" → 10)
def clean_ratings(r):
    if isinstance(r, str):
        match = re.search(r"\d+", r)
        return int(match.group()) if match else 0
    return 0

df_copy["Number of Ratings"] = df_copy["Number of Ratings"].apply(clean_ratings)
```

### Polars

```
# b. Replace NaN in specific columns with "unknown"
df_lazy = df_lazy.with_columns([
    pl.col("Product Name").fill_null("unknown"),
    pl.col("Location").fill_null("unknown"),
    pl.col("Price").fill_null("unknown")
])
```

```
# e. Clean "Number of Ratings" (e.g., "(10)" → 10)
df_lazy = df_lazy.with_columns([
    pl.col("Number of Ratings")
    .cast(pl.Utf8)
    .str.extract(r"(\d+)")
    .cast(pl.Int64)
    .fill_null(0)
    .alias("Number of Ratings")
])
```

# Polars

## Performance of Polars

### Pandas

```
⏱️ Elapsed Time: 9.12 sec
📊 Memory Used (Start → End): 248.86 MB → 315.95 MB
🚀 Peak Memory (tracemalloc): 41.63 MB
📈 Throughput: 12,758.67 records/sec
📄 Total Records Cleaned: 116308
```

### Polars

```
⏱️ Elapsed Time: 0.82 sec
📊 Memory Used (Start → End): 438.77 MB → 530.44 MB
🚀 Peak Memory (tracemalloc): 26.82 MB
📈 Throughput: 142,084.92 records/sec
📄 Total Records Cleaned: 116308
```

# PySpark

## Why choose Pyspark?

### Scalability

PySpark handles large datasets distributed across clusters, while Pandas works best with data that fits into a single machine's memory.

### Parallel Processing

PySpark processes data in parallel on multiple nodes, significantly speeding up computations; Pandas runs mostly single-threaded.

### Optimized Execution

PySpark uses Catalyst optimizer and Tungsten engine to optimize queries and resource use, Pandas lacks such optimizations.

# PySpark

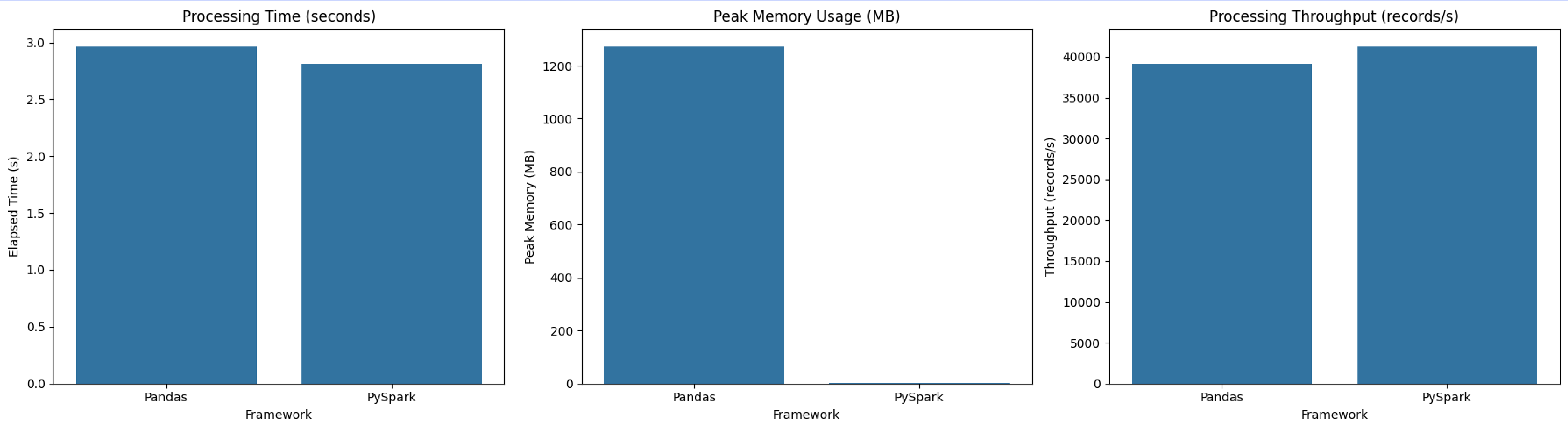
## Code Comparison

```python
# d. Clean "Quantity Sold" (e.g., "5K sold" → 5000)
def clean_quantity(q):
    if isinstance(q, str):
        q = q.lower().replace("sold", "").strip()
        if "k" in q:
            return int(float(q.replace("k", "")) * 1000)
        return int(re.findall(r"\d+", q)[0]) if re.findall(r"\d+", q) else 0
    return 0

df_copy.loc[:, "Quantity Sold"] = df_copy["Quantity Sold"].apply(clean_quantity)
```

```python
# 4.Clean "Quantity Sold" (e.g., "5K sold" → 5000)
sdf = sdf.withColumn(
    "Quantity Sold",
    when(
        col("Quantity Sold").rlike(".*K.*"),
        (regexp_replace(col("Quantity Sold"), "K", "").cast("int") * 1000)
    ).otherwise(col("Quantity Sold").cast("int"))
)
```

# PySpark

## Performance Comparison




## Pandas

⏱ Elapsed Time: 2.97 sec
📊 Memory Used (Start → End): 686.22 MB → 603.24 MB
🚀 Peak Memory (tracemalloc): 126.92 MB
📈 Throughput: 39,172.01 records/sec
📄 Total Records Cleaned: 116296

## PySpark

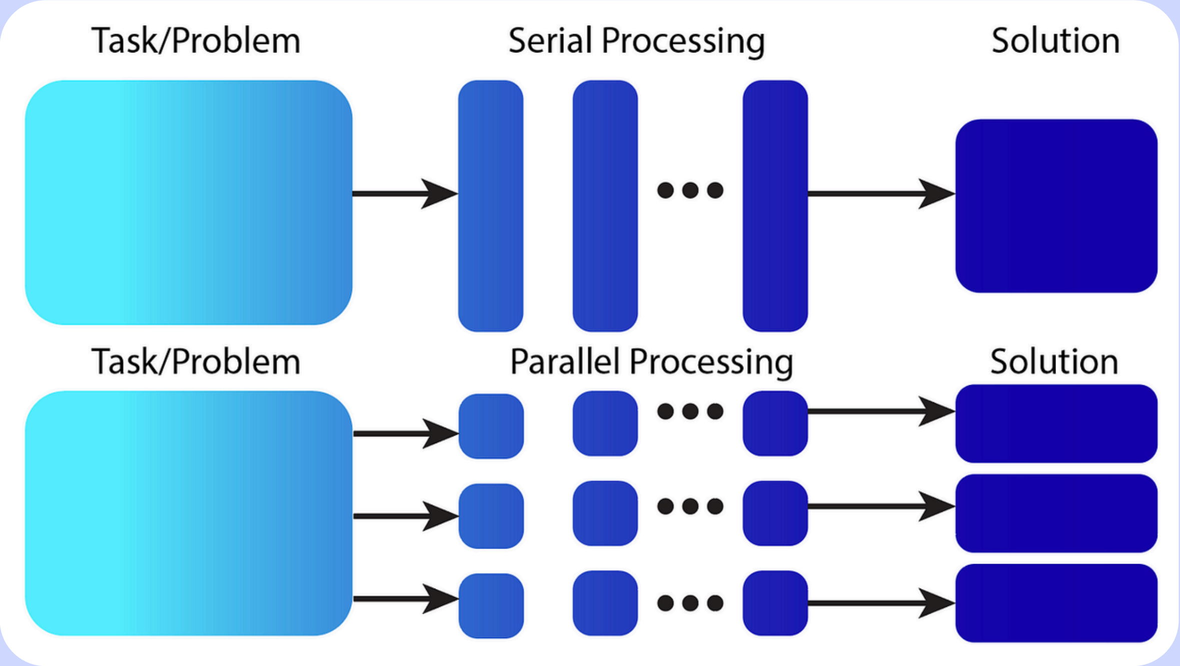
⏱ Elapsed Time: 2.81 sec
📊 Memory Used (Start → End): 571.34 MB → 571.34 MB
🚀 Peak Memory (tracemalloc): 0.09 MB
📈 Throughput: 41,322.38 records/sec
📄 Total Records Cleaned: 116296

# Dask

## How does Dask optimize the Cleaning Process?

### Lazy Evaluation via Task Graph

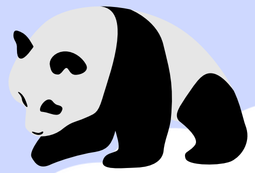
### Out-of-Core and Blockwise Parallelism




### Scheduler-Based Parallel Execution

# Dask

## Data Cleaning Steps Comparison



**Pandas**

**dask**

### Clean "Quantity Sold"

Pandas:

```
df["Quantity Sold"] = df["Quantity Sold"].apply(clean_quantity)
```

Dask:

```
df = df.assign(
    Quantity_Sold=df["Quantity Sold"].map(clean_quantity, meta=("Quantity_Sold", "int64"))
)
```

### Fill Missing Value

Pandas:

```
df.fillna({
    "Product Name": "unknown",
    "Location": "unknown",
    "Price": "unknown",
    "Quantity Sold": "0",
    "Number of Ratings": "0"
}, inplace=True)
```

Dask:

```
df = df.fillna({
    "Product Name": "unknown",
    "Location": "unknown",
    "Price": "unknown",
    "Quantity Sold": "0",
    "Number of Ratings": "0"
})
```

### Trigger Execution

Pandas:

⛔ Not required — all operations already executed eagerly.

Dask:

```
df_clean_dask = df.compute()
```

# Dask

## Performance Comparison

### Pandas

⏱ Elapsed Time: 8.84 sec
📊 Memory Used (Start → End): 512.98 MB → 544.23 MB
🚀 Peak Memory (tracemalloc): 39.11 MB
📈 Throughput: 13,162.45 records/sec
📄 Total Records Cleaned: 116308

### Dask

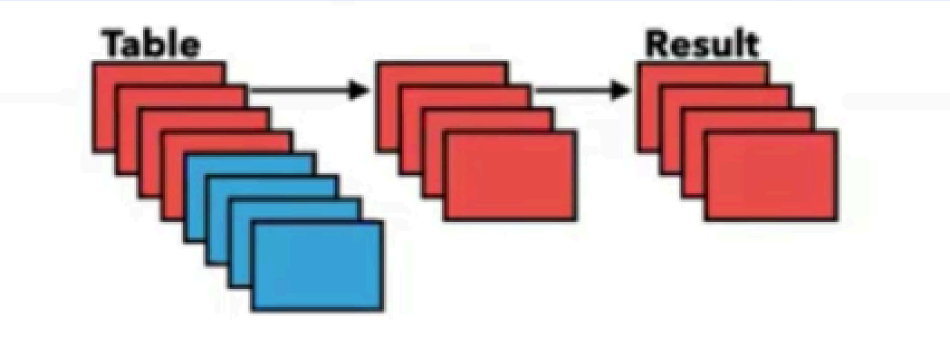
⏱ Elapsed Time: 9.61 sec
📊 Memory Used (Start → End): 298.34 MB → 314.79 MB
🚀 Peak Memory (tracemalloc): 58.65 MB
📈 Throughput: 12,097.35 records/sec
📄 Total Records Cleaned: 116308

# DuckDB

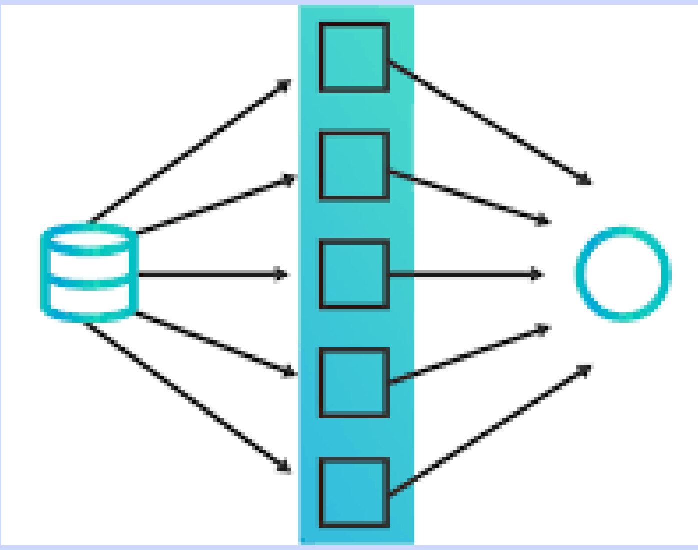
## How DuckDB Optimize The Process ?

### Columnar Engine


### Vectorized Execution

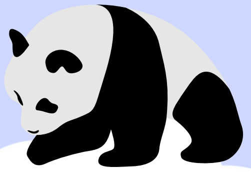



### Multithreading

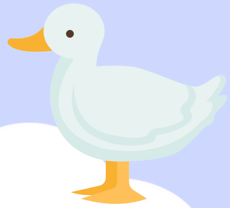


# DuckDB

## Data Cleaning Steps Comparison



### Pandas



### DuckDB

### Drop Duplicates

Pandas:

```
df.drop_duplicates(inplace=True)
```

DuckDB:

```
SELECT DISTINCT * FROM
```

### Fill Missing Value

Pandas:

```
df_copy["Product Name"].fillna("unknown")
```

DuckDB:

```
WHEN "Product Name" IS NULL OR "Product Name" = ''
THEN 'unknown'
ELSE "Product Name"
```

### Convert Data Type

Pandas:

```
df_copy["Number of Ratings"].astype(int)
```

DuckDB:

```
CAST(regexp_extract("Number of Ratings", '\\d+') AS INT)
```

# DuckDB

## Performance Comparison

### Pandas

⏱️ Elapsed Time: 8.84 sec
📊 Memory Used (Start → End): 512.98 MB → 544.23 MB
🚀 Peak Memory (tracemalloc): 39.11 MB
📈 Throughput: 13,162.45 records/sec
📄 Total Records Cleaned: 116308

### DuckDB

⏱️ Elapsed Time: 4.79 sec
📊 Memory Used (Start → End): 517.74 MB → 515.71 MB
🚀 Peak Memory (tracemalloc): 31.88 MB
📈 Throughput: 24,265.28 records/sec
📄 Total Records Cleaned: 116296

# Performance Evaluation

| Framework | Average Time (sec) | Peak Memory (MB) | Throughput (records/sec) | Strengths | Use Case Fit |
|---|---|---|---|---|---|
| Pandas | 8.84 | 41.16 | 13,167 | Simple, in-memory speed, ease of use | Small to medium datasets (<1M rows) |
| Polars | 0.83 | 26.80 | 140,433 | Columnar, lazy execution, fast in-memory ops | Fastest single-node processing |
| PySpark | 2.81 | 0.09 | 41,322 | Distributed, cluster-scale parallelism, built for big data | Large-scale or cluster environments |
| Dask | 9.61 | 58.65 | 12,097 | Parallelism, Pandas-like syntax, chunked memory | Larger-than-mem datasets |
| DuckDB | 5.51 | 31.87 | 21,124 | In-process OLAP engine, zero-copy, SQL-style queries | Analytical queries, efficient local |

# Performance Evaluation

## Comparative Analysis

- Polars proved to be the fastest in execution due to its Rust-based engine and lazy evaluation.
- DuckDB showed strong performance using vectorized and SQL-style processing, with most memory-efficient due to its in-process engine and vectorized queries.
- PySpark offered the best scalability for distributed, large-scale workloads but added overhead for smaller datasets.
- Dask is better suited for larger or partitioned datasets, stood out for its flexibility and compatibility with the familiar Pandas API and parallel processing.
- Pandas remained the most beginner-friendly tool, thanks to its intuitive syntax, extensive documentation, widespread use, reliable and quick for the given dataset size, but not optimized for scaling.

# Performance Evaluation

## Performance Ranking

- 1st place: Polars leads the fastest runtime with its high throughput, and low memory use.
- 2nd place: PySpark offered excellent throughput, high scalability, and minimal memory usage.
- 3rd place: DuckDB showed balanced performance with strong memory efficiency and throughput.
- 4th place: Pandas which is suitable for small to medium data and offers moderate performance.
- 5th place: Dask is best for scalability beyond memory, but the slowest one for this dataset size of approximately ~116,000 rows of data with 5 columns.

# Challenges and Limitation

- CAPTCHA & Bot Prevention
- Missing Product Data
- Dynamic JavaScript Content
- Scalability Issues
- Limited & Unreliable Pagination
- Search Algorithm Restrictions

# Thank You