Stream and Batch Processing of Water Quality

# Introduction

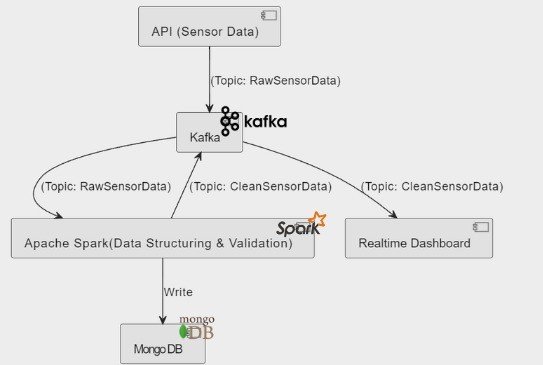
This project develops a real-time data pipeline utilizing Apache Kafka and Apache Spark to handle and analyze water quality data. The primary goal is to demonstrate eIective real-time data processing compared to traditional batch processing, using simulated sensor data to model water quality metrics.

# Problem Description

Water quality monitoring is vital for environmental health and public safety. Traditional batch processing methods can delay the action needed to address pollution or contamination events. This project explores a real-time processing solution to provide immediate insights into water quality data.

# Solution Architecture

* + API: Simulates water quality sensor data, mirroring actual sensor outputs.
  + Streaming Apps/Tools: Uses Apache Kafka for real-time data ingestion and Apache Spark for on-the-Ry data processing.
  + Repository/Database: MongoDB for data persistence post-analysis, enabling further historical analysis.



Architecture Diagram

# Installation of Software

* **Apache Kafka (3.7.0)** : a distributed streaming platform that makes it possible to create streaming apps and real-time data pipelines. We utilized this as a publisher and for data streaming.

<https://downloads.apache.org/kafka/3.7.0/kafka_2.12-3.7.0.tgz>

* **Apache Spark Streaming (3.5.1)** : Built on top of the Apache Spark platform, a real-time processing engine enables real-time data stream processing. permits batch processing as well.

<https://www.apache.org/dyn/closer.lua/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz>

* **Pyspark (3.5.1) :** A streaming engine designed speci1cally for Python and its related features, similar to Apache Kafka.

TO INSTALL : pip install pyspark==”3.5.1”

* **DBMS Used :** MongoDB - 7.0.8 <https://www.mongodb.org/static/pgp/server-7.0.asc>

# Streaming Apps/Tools Used

## Apache Kafka:

Apache Kafka facilitates the publishing and subscribing of streams of records, which enables real-time data processing. Kafka provides fault tolerance, scalability, and high throughput, making it suitable for building real-time data pipelines.

## Apache Spark Streaming:

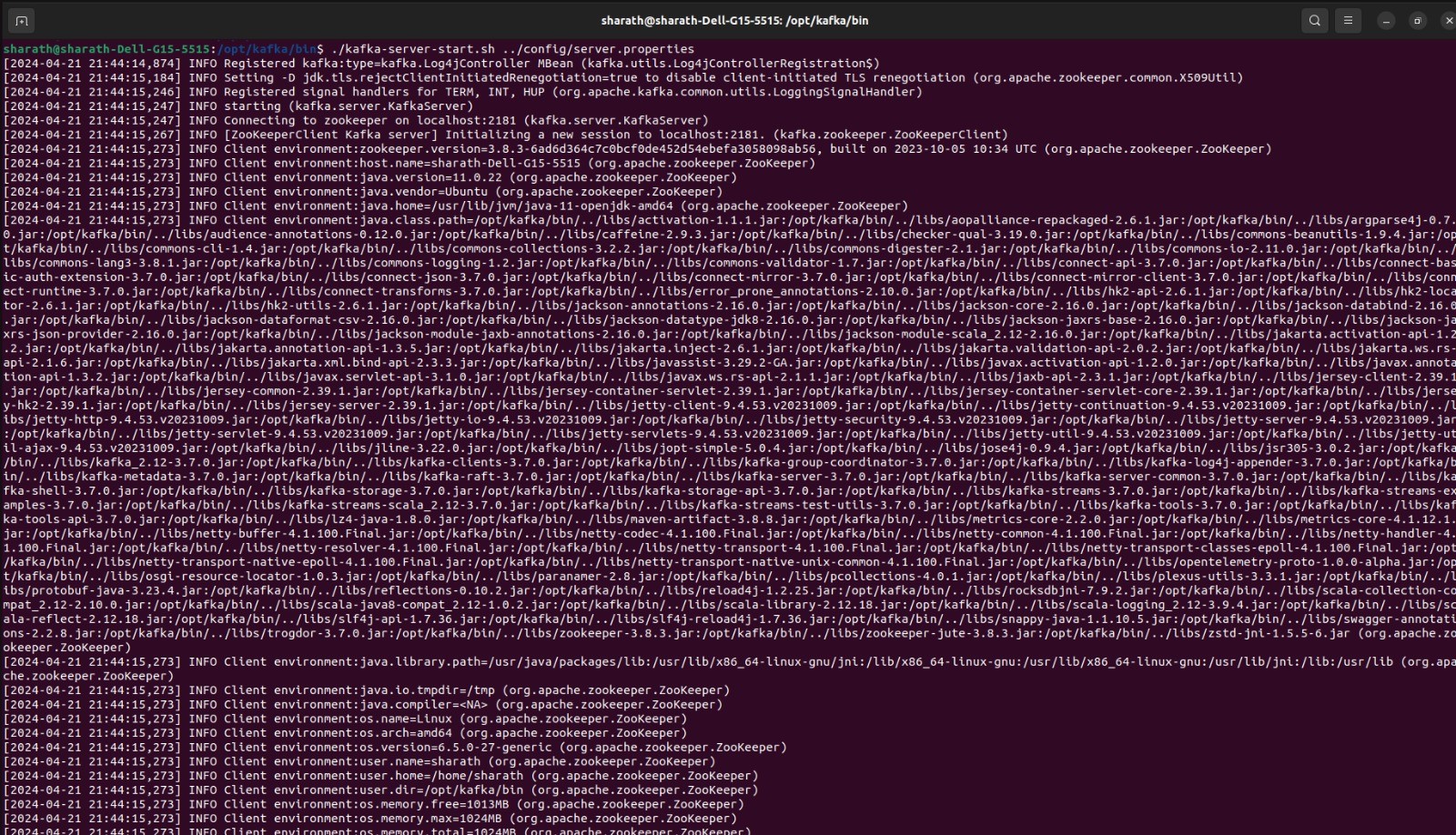
Apache Spark Streaming is used for processing real-time data streams from Kafka. It allows for scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming enables the application of complex algorithms and analytics on real-time data.

## MongoDB:

MongoDB is a NoSQL database used for storing real-time sensor data processed by Spark Streaming.

## Zookeeper:

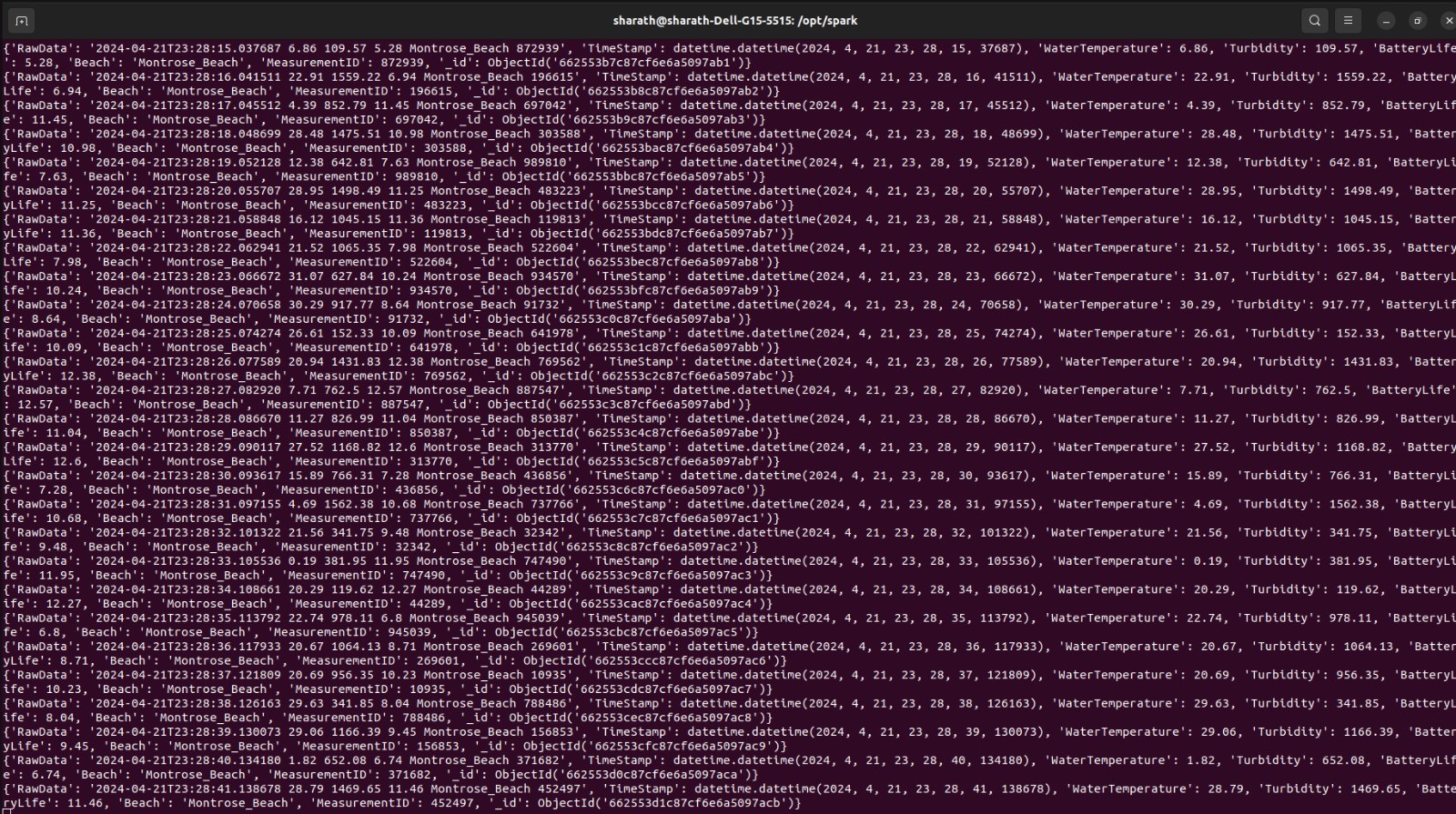
In this project, Zookeeper is used for managing con1gurations and coordination in the Kafka ecosystem.



The above 1gure represents starting the Kafka server using the speci1ed con1guration 1le.

# Database

Mongodb is utilized in our project as a NoSQL database, which stores data in JSON-like documents with dynamic schemas (BSON), making the integration of data in certain types of applications easier and faster. In our project, MongoDB stores real-time sensor data, such as water temperature, in a collection within the 'RealTimeDB' database. This approach enables e cient data storage and fast retrieval capabilities, essential for performing real-time analytics and batch processing. The use of aggregation queries in MongoDB facilitates complex analytics, such as calculating average temperatures from accumulated data, directly within the database, thereby optimizing performance and scalability.



The above 1gure represents the data stored in Mongodb after streaming.

# Kafka Topics

## RawSensorData:

This topic is used to store raw sensor data obtained from the API. Data published to this topic

includes information such as timestamp, water temperature, turbidity, battery life, beach, and measurement ID. Raw sensor data is streamed into this topic by the push\_data\_to\_kafka.py script.

## ValidatedSensorData:

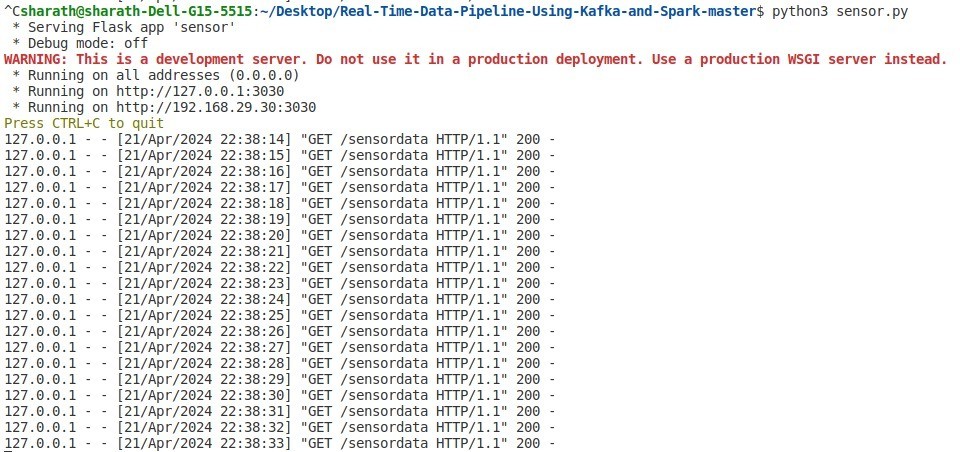
This topic acts as an intermediate step for validated sensor data before it's persisted into MongoDB. After the sensor data is received from the RawSensorData topic, it undergoes validation to ensure its integrity and correctness. Validated sensor data is published to this topic by the structure\_validate\_store.py script.

## CleanSensorData:

This topic stores the cleaned and validated sensor data after it's persisted into MongoDB. Once the data is validated and stored in MongoDB, it's published to this topic by the structure\_validate\_store.py script. Cleaned sensor data from this topic can be consumed by downstream systems, such as real-time dashboards, for visualization or further analysis.

# Input Data

1. **Sources :** Beach water quality API from the sensors in the water at beaches along Chicago’s lake Michigan

<https://data.world/cityofchicago/beach-water-quality-automated-sensors>

1. **Description :** The Chicago Park District maintains sensors in the water at beaches along Chicago's Lake Michigan lakefront. These sensors generally capture the indicated measurements hourly while the sensors are in operation during the summer. During other seasons and at some other times, information from the sensors may not be available. See https://data.cityofchicago.org/d/k7hf-8y75 for a dataset with land-based weather measurements at selected beaches. The sensor locations are listed at https://data.cityofchicago.org/d/g3ip-u8rb. Please note that sensor locations change with the Park District’s operational needs, primarily related to water quality.

# Streaming Mode Experiment

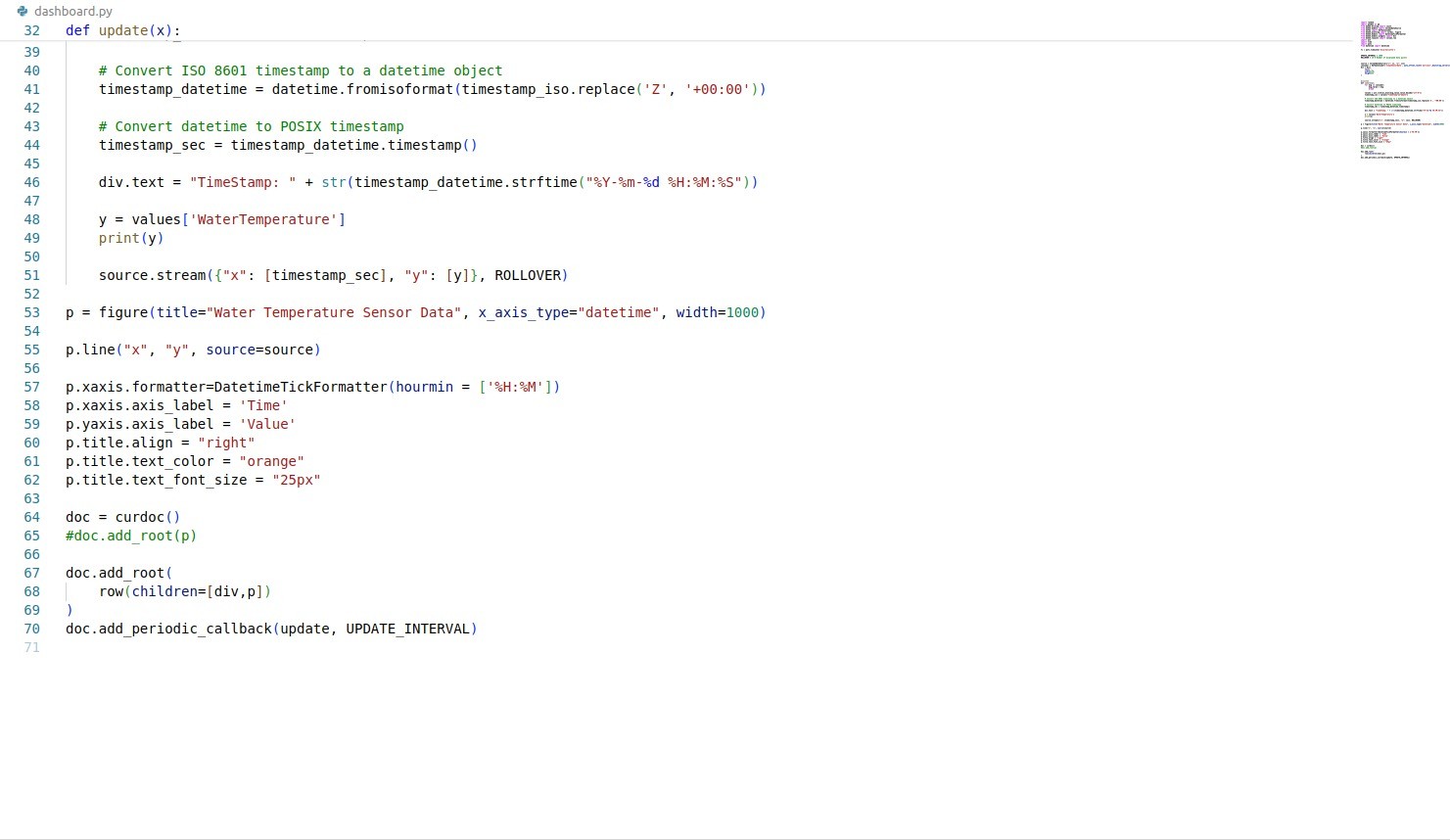
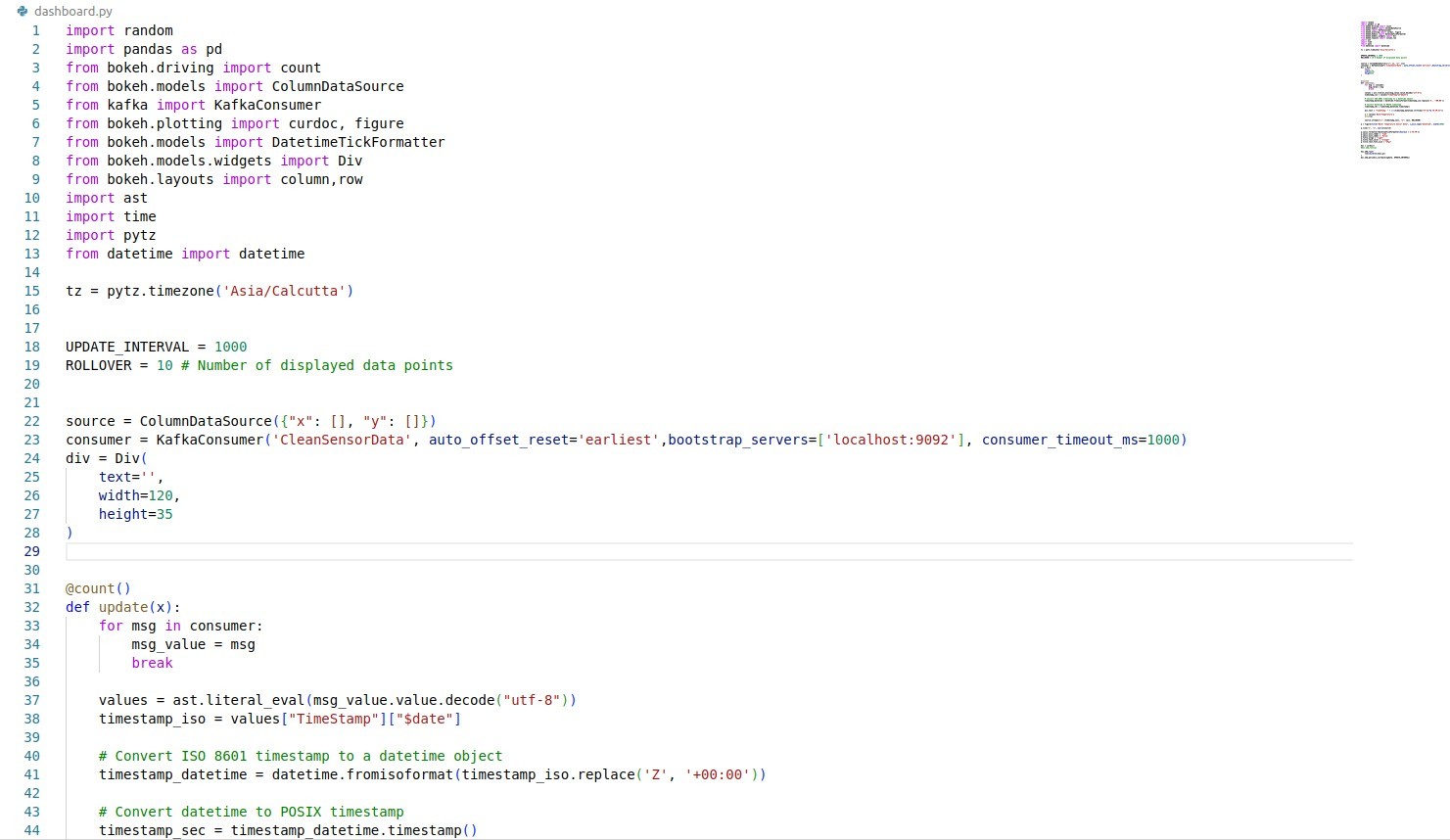
1. **Description :** This experiment is designed to demonstrate the system's ability to handle and analyze continuous data streams using a combination of Apache Kafka and Apache Spark Streaming. Kafka serves as the initial ingestion point, reliably capturing live data streams that simulate environmental sensor outputs such as water temperature and turbidity. Once ingested, data is forwarded to Spark Streaming, which is con1gured to process this data in real time. The integration of these two technologies ensures seamless data Row and immediate processing, which is crucial for scenarios requiring rapid analytical feedback and decision-making.

# Windows :

**Time Windows**: The streaming data is processed using a windowing technique, where data is grouped into 1xed windows of one minute each. This approach allows the system to perform time-based aggregations and analyses, such as calculating the average temperature or detecting outliers and anomalies within each window. The use of windowing not only simpli1es the processing of continuous streams by breaking down the in1nite stream into manageable chunks but also enables the application of complex analytical models on a per-window basis.

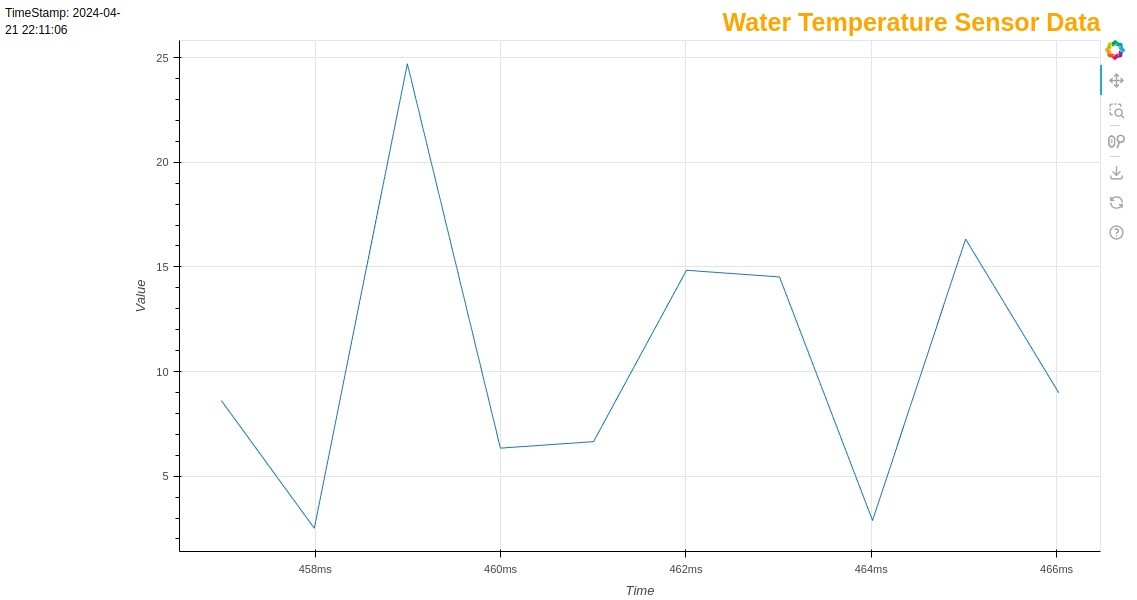
**Window Operations**: Within each window, Spark Streaming executes several functions—aggregating temperature data to compute averages and applying statistical

models to detect sudden changes that could indicate environmental anomalies. These operations are critical for applications like environmental monitoring, where timely detection of data trends and anomalies can lead to prompt action to mitigate potential hazards.



This is dashboard.py. It shows how to consume data from Kafka and prepare it for real-time visualization

# Results :



The provided graph represents the real-time Ructuations in water temperature as captured and processed by the streaming system. The X-axis indicates the time, seemingly in milliseconds, while the Y-axis represents the water temperature values.

**Real-Time Visualization**: The outcomes of the streaming data analysis are dynamically visualized using a real-time dashboard. This visualization includes trend lines for water temperature over time, highlighted points where anomalies are detected, and alerts that are triggered based on speci1c threshold conditions.

System Responsiveness: The experiment highlights the system's responsiveness, as changes in the data stream are quickly reRected in the visualizations. For instance, a sudden spike in temperature due to an environmental factor is immediately captured, processed, and displayed, showcasing the potential of the system to support real-time environmental monitoring.

Potential for Immediate Environmental Monitoring: The ability to monitor environmental conditions in real time has signi1cant implications. For industries such as

water quality management, having immediate access to data about changes in water conditions can be crucial for pollution control and ecological conservation. The system’s capability to provide continuous and instant analysis makes it an invaluable tool in such contexts.

# Batch Mode Experiment

1. **Description :**

The batch mode experiment in this project is designed to process accumulated data stored in MongoDB using Apache Spark, one of the leading platforms for large-scale data processing. This setup allows for periodic processing of large data batches, which are structured similarly to the real-time streams but are processed in intervals (e.g., daily or hourly). By performing the same types of aggregations as those done in the streaming experiment, such as calculating average temperatures or detecting outliers, the batch mode experiment provides a basis for directly comparing performance and outcomes between the two processing modes.

Spark Integration with MongoDB: Apache Spark is con1gured to read data directly from MongoDB using the MongoDB Connector for Spark. This integration facilitates seamless data transfer into Spark’s dataframes, which are then used for further analysis.

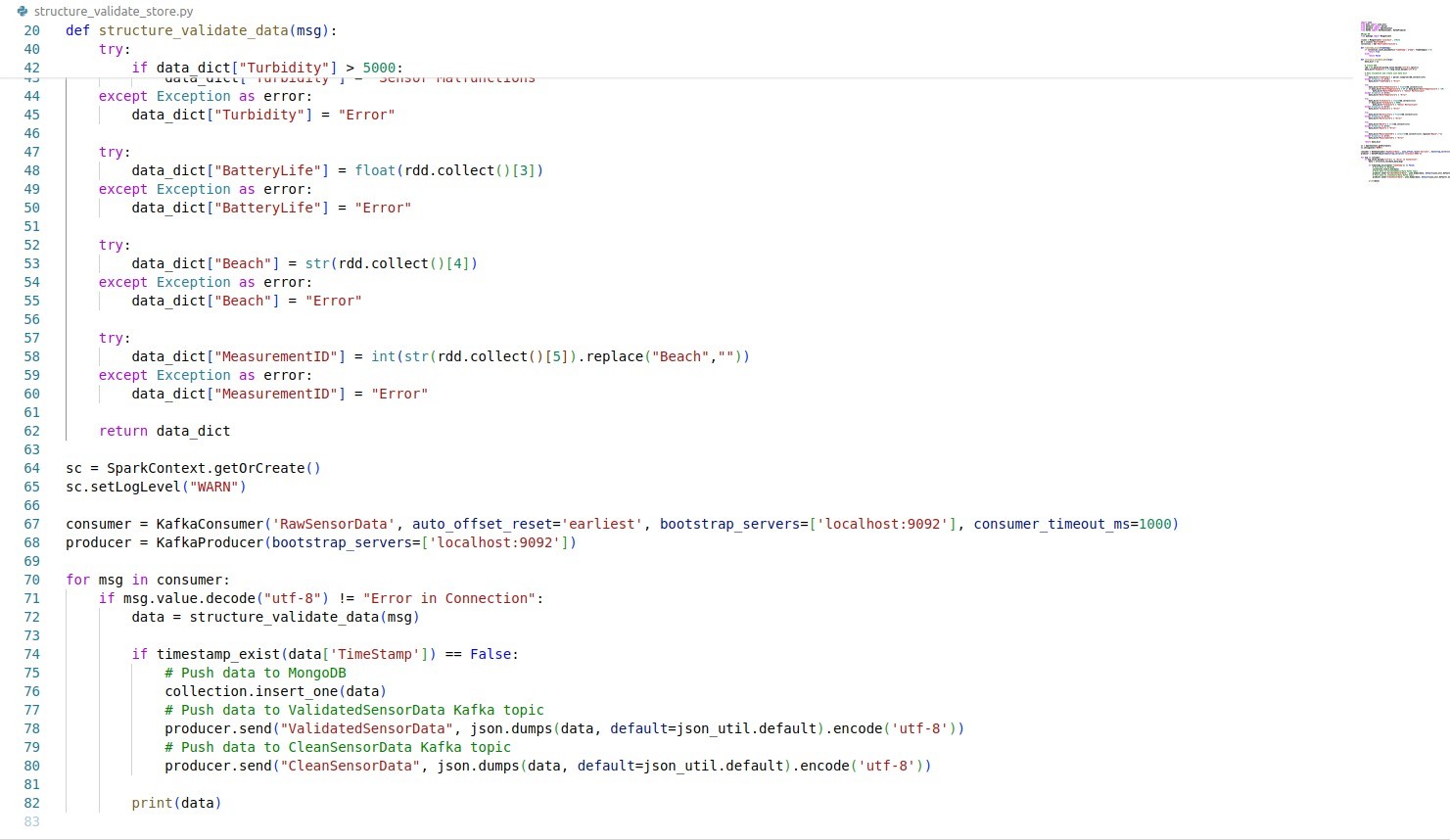
Data Aggregation and Analysis: Spark employs sophisticated data processing algorithms to e ciently aggregate and analyze data. The experiment focuses on calculating average temperatures and other statistical metrics from sensor data, leveraging Spark’s ability to handle complex computations over large datasets.

# Data Size :

**Handling Large Datasets:** The batch processing experiment is speci1cally designed to manage and process batches containing large records. This scale is chosen to demonstrate Spark’s capabilities in handling substantial volumes of data without compromising on processing speed or accuracy.

**Data Batching Strategy:** The data is grouped into batches based on either time intervals or record count, ensuring consistent and manageable batch sizes for processing. This strategy helps in optimizing the processing workload and aligning it with the system’s

capacity.



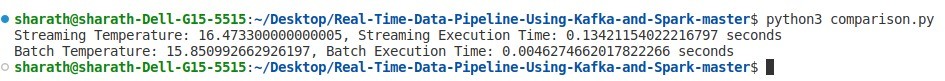
This is structure\_validate\_store.py.

# Result :

The batch mode processing was performed on a dataset using Apache Spark to calculate aggregated metrics, such as the average water temperature. The results of the execution yielded an average water temperature of 15.85°C. This 1gure is representative of the environmental conditions within the timeframe of the accumulated data. The calculation was executed in a notably rapid execution time of 0.0040602 seconds, demonstrating the batch processing's capability for high-speed computation on stored datasets.

The quick execution time is particularly noteworthy, suggesting that the Spark-based batch processing system is well-optimized for such aggregative computations. Furthermore, this fast processing capability of batch operations, when applied to large volumes of data that do not require real-time analysis, could oIer organizations a time-e cient and cost-eIective solution for periodic data analysis tasks.

# Comparison of Streaming & Batch Modes



## Temperature Results Comparison:

The average temperatures obtained from the streaming and batch modes are relatively close, with a minor diIerence that could be attributed to the datasets' time frame or variations in data sampled during the streaming window versus the entire batch dataset.

Streaming Mode: Produced an average temperature of 16.47°C, reRecting the near-instantaneous analysis of the data as it streams in real time.

Batch Mode: Yielded an average temperature of 15.86°C, indicating the processing of accumulated data over a given period.

## Execution Time Comparison:

There is a signi1cant diIerence in execution times between the two modes, which highlights the trade-oIs between real-time processing and batch data analysis.

Streaming Mode: Demonstrates a processing time of approximately 0.13 seconds. This duration is reRective of the near-real-time requirement for streaming data processing, where data is continually ingested and processed in small windows, necessitating quick turnarounds.

Batch Mode: Showcases a faster execution time of 0.005 seconds. This e ciency can be largely attributed to the processing of a predetermined dataset in one go, allowing for more optimized computation and resource allocation.

## Discussion:

Timeliness: Streaming processing is designed to provide ongoing insights, which is crucial for time-sensitive decisions. The slightly higher average temperature in streaming could indicate a real-time response to a transient environmental condition.

Resource E ciency: Batch processing appears to be more resource-e cient in this instance, handling a large dataset with a quicker execution time than streaming. This suggests that for tasks where real-time analysis is not critical, batch processing could be more favorable in terms of both speed and computational load.

Use Case Suitability: The choice between streaming and batch processing should be informed by the speci1c use case requirements. For operations that rely on the most up-to-date information, such as fraud detection or emergency response systems, streaming oIers a distinct advantage. Conversely, for analytical tasks that can be scheduled and do not require immediate action, batch processing is advantageous due to its e ciency and speed.

# Conclusion:

The project demonstrates the bene1ts of real-time data processing for quick water quality monitoring, while also showing how batch processing is useful for detailed analysis. This

helps inform decisions on setting up data processing systems for water quality monitoring.

# References

**a. URLs**

1. <https://downloads.apache.org/kafka/3.7.0/kafka_2.12-3.7.0.tgz>
2. [https://www.apache.org/dyn/closer.lua/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tg](https://www.apache.org/dyn/closer.lua/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz) [z](https://www.apache.org/dyn/closer.lua/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz)
3. <https://www.mongodb.org/static/pgp/server-7.0.asc>
4. <https://data.world/cityofchicago/beach-water-quality-automated-sensors>
5. <https://data.cityofchicago.org/d/k7hf-8y75>
6. [https://data.cityofchicago.org/d/g3ip-u8rb.](https://data.cityofchicago.org/d/g3ip-u8rb)
7. [https://www.youtube.com/watch?v=udnX21 SuU&ab\_channel=LearningJournal](https://www.youtube.com/watch?v=udnX21__SuU&ab_channel=LearningJournal)
8. <https://www.youtube.com/watch?v=QaoJNXW6SQo&ab_channel=Simplilearn>