DSP Project: Weather Data Analysis and Kafka Integration

Alpha Oumar Barry, Vianney Grimaud

First Semester 2023-2024

Abstract

Abstract

This project aims to fetch real-time weather data from WeatherAPI.com, produce it to Kafka, and perform temperature analysis. The project is implemented in Python and uses Kafka for real-time data streaming, Streamlit for web application development, and other packages for geographical data plotting. The report provides an overview of the project's objectives, methodology, implemented models, experimental results, and conclusions. A demonstration of the code's functionality is included, and contributions from contributors are acknowledged. The project is available on Github.

Contents

1 Introduction

The Weather Data Analysis and Kafka Integration project is a comprehensive solution designed to meet the growing need for efficient and scalable real-time weather data processing. This report provides an in-depth overview of the project, detailing the construction of a Streamlit dashboard for visualizing weather data, the integration with Kafka for seamless data processing, and the exploration of temperature trends through various analysis techniques.

2 Problematic and Methodology

The primary objective of this project is to construct a user-friendly dashboard that allows users to explore real-time weather data in an intuitive manner. To achieve this, the project utilizes the WeatherAPI.com API to fetch current weather information, Apache Kafka for efficient data streaming, and employs various analysis techniques to gain insights into temperature trends.

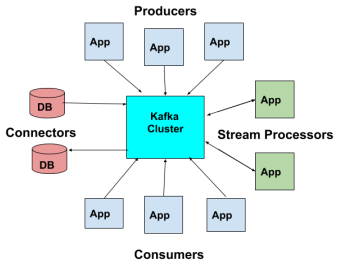


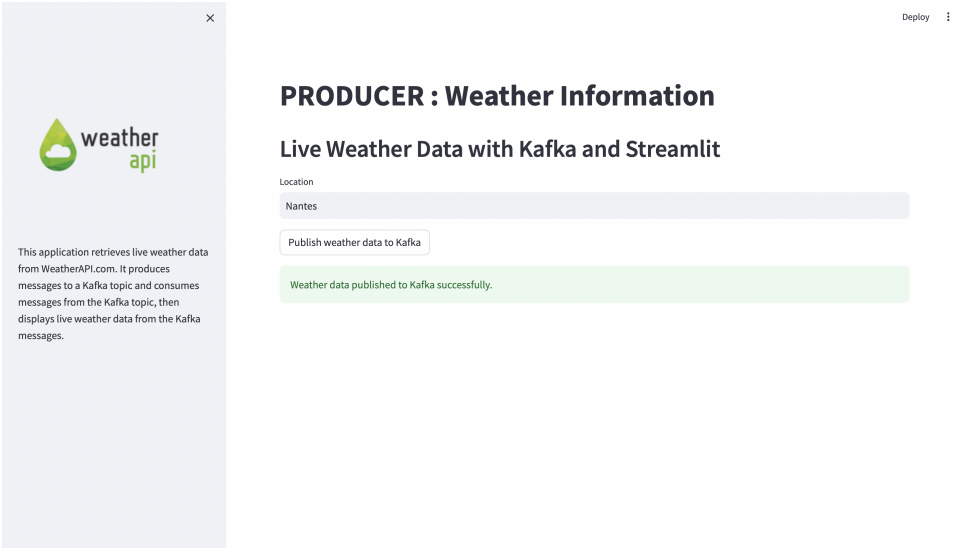
Figure 1: A schematic representation of the Kafka integration for real-time weather data processing and visualization.

2.1 Dashboard Construction

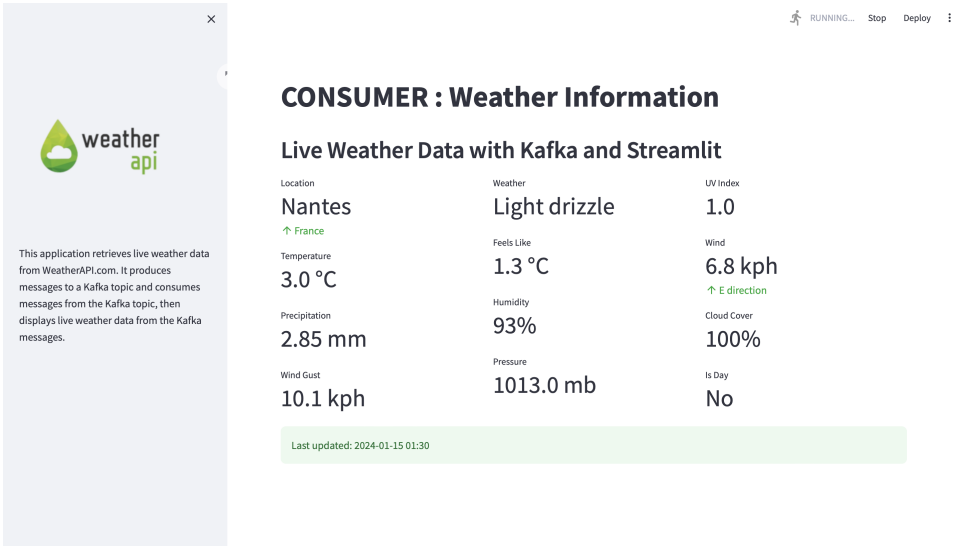
The Streamlit dashboard, implemented in the `dashboard.py` script, provides an interactive platform for users to explore weather data. It offers features such as searching for weather information by location, displaying current conditions. The dashboard is designed with a focus on user experience, ensuring that even complex weather data is presented in an easily understandable format.

2.2 Kafka Integration

The `kafka_producer.py` script fetches real-time weather data for a specified location and publishes it to a Kafka topic. The script uses the KafkaProducer class from the kafka Python library to create a Kafka producer. The `publish_messages_to_kafka` function is responsible for fetching the weather data and sending it to the Kafka topic. The weather data is fetched using the `fetch_weather_data` function, which presumably makes a request to a weather API and returns the weather data for the specified location. If the weather data is successfully fetched, it is published to the Kafka topic using the send method of the Kafka producer.



The `kafka_consumer.py` script consumes the weather data from the Kafka topic and displays it on a Streamlit dashboard. The script uses the KafkaConsumer class from the kafka Python library to create a Kafka consumer. The `consume_messages_from_kafka` function is responsible for consuming the messages from the Kafka topic. Each message is passed to the `handle_weather_info` function, which extracts the weather data from the message and updates the Streamlit dashboard with the new data.



This setup allows for real-time processing and visualization of weather data. As new weather data is published to the Kafka topic, it is immediately consumed by the Kafka consumer and displayed on the Streamlit dashboard. This ensures that the dashboard always displays the most up-to-date weather information. The use of Kafka also provides scalability, as it can handle high volumes of data and allows for multiple consumers to consume the data simultaneously. This makes it an ideal solution for real-time applications that require efficient data streaming.

3 Extended Analysis and Experimentation

The analysis scripts demonstrate a powerful approach to real-time weather data analysis, for example, through online machine learning with the `river` library. In this analysis, we explore two additional components of the script, each addressing distinct aspects of real-time data streaming and historical data analysis. The first component involves streaming real-time weather data for visual representation, while the second component delves into historical data analysis and regression modeling.

3.1 Streaming Real-time Weather Data on a Map

The first analysis code `kafka_mapping.py` focuses on streaming real-time weather data for visual representation. The script employs various libraries, including `json`, `requests`, `matplotlib` and `Kafka`, to create an interactive and dynamic map visualizing the current weather conditions in France. The script fetches real-time weather data from an external API using the `WEATHER_API_KEY`, and Kafka is utilized for data streaming.

A Kafka producer is created to send real-time weather data to a specified Kafka topic. This data is then consumed by a Kafka consumer, which processes the information and up-

dates a map of France. The map color-codes regions based on temperature, providing a dynamic visualization of temperature variations across different geographic areas. Additionally, we could have displayed weather icons on the map, enhancing the visual representation of current conditions (but we went back because of the readability objective of the map).

This streaming feature enhances the script's utility, providing a real-time and interactive tool for monitoring weather patterns. The animated map dynamically adapts to incoming data, offering a comprehensive view of temperature variations and conditions across France.

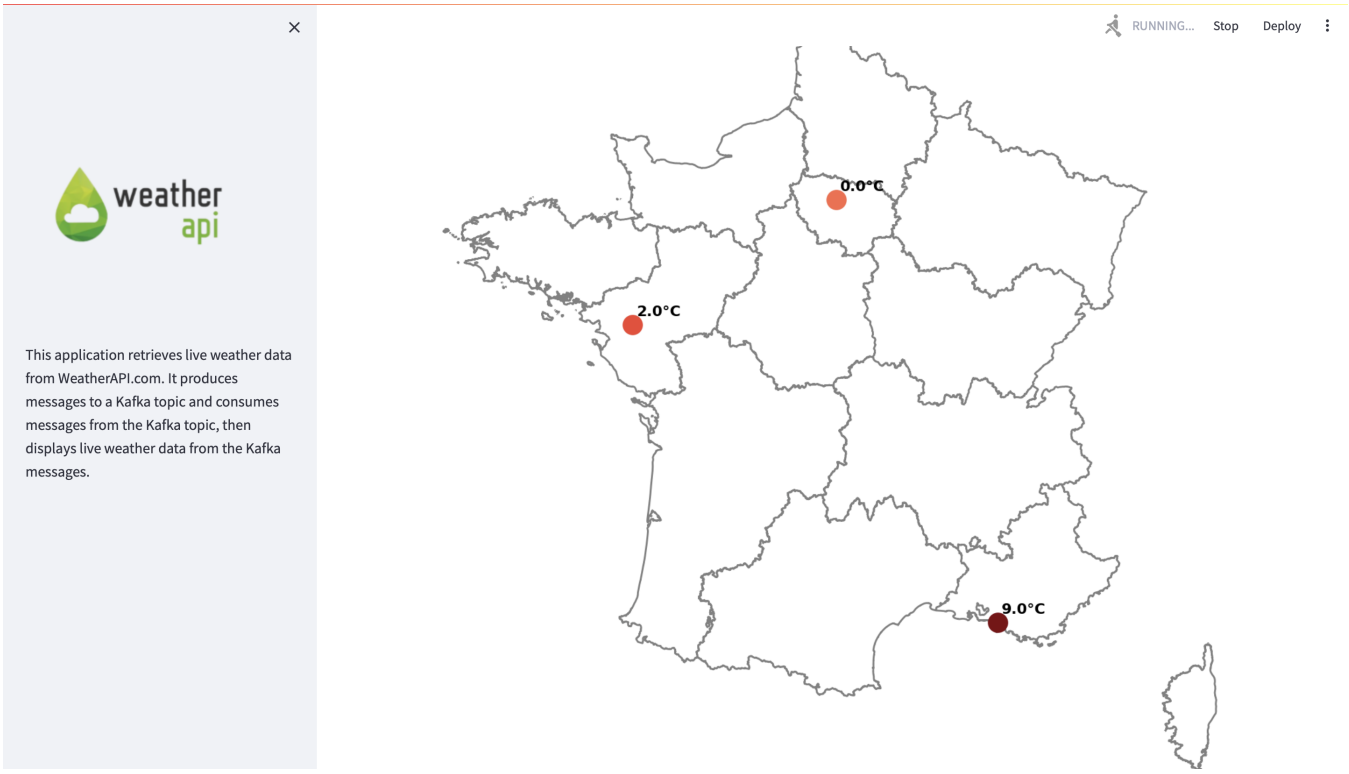


Figure 2: A screen of the Kafka integration for real-time weather on the french map.

3.2 Historical Data Analysis and Regression Modeling

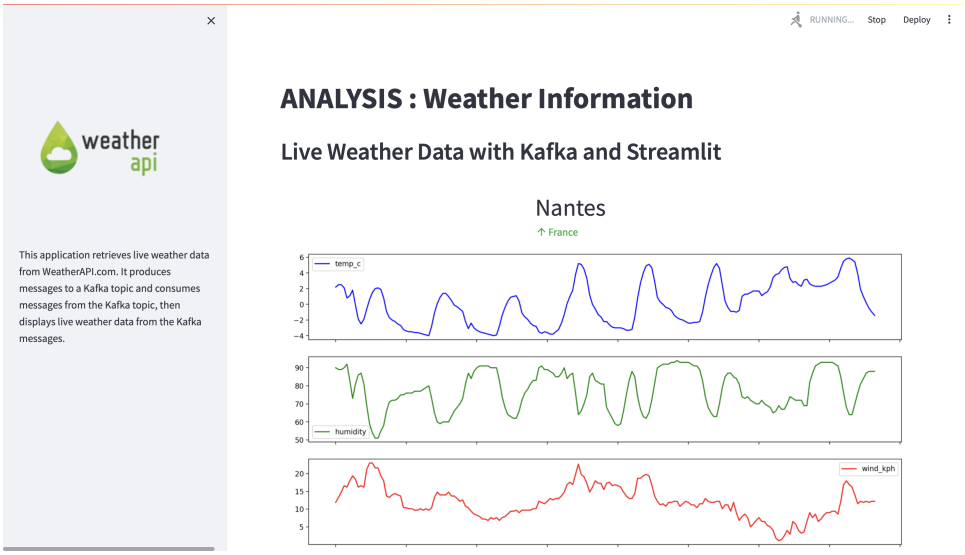
The second analysis code `kafka_regression.py` extends the script's capabilities by incorporating historical data analysis and regression modeling. This segment utilizes libraries such as `pandas`, `matplotlib`, `sklearn`, `requests`, and `json` to fetch historical weather data for a specified location and perform regression analysis.

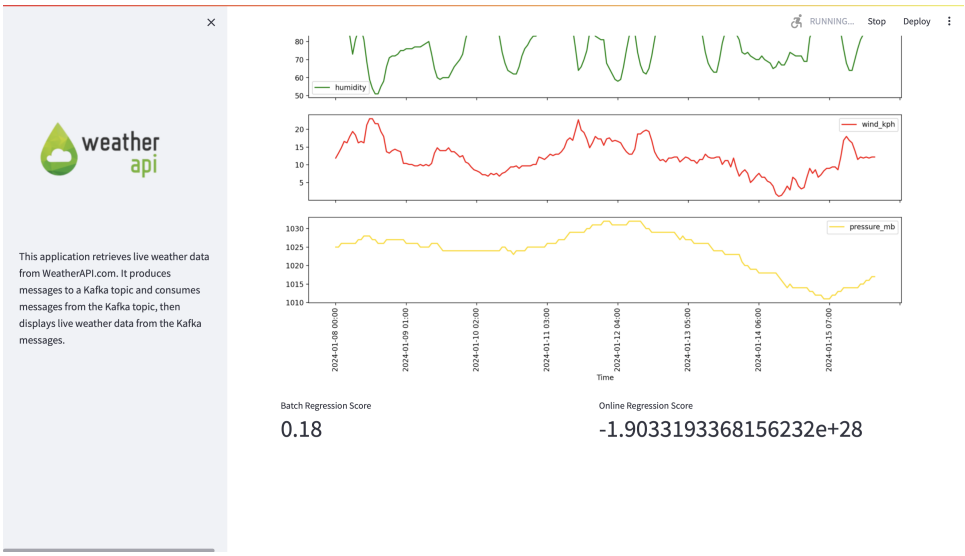
The script fetches historical weather data for a given location and date range (limited because of the free subscription to the API), processing the data to create a dataframe. The variables of interest, including temperature, humidity, wind speed, and pressure, are plotted

over time. This visual representation allows for a comprehensive exploration of historical weather patterns.

The script then performs regression analysis using both batch and online learning models. The `scikit-learn` library is leveraged to create a linear regression model for batch learning, while the `river` library's stochastic gradient descent (SGD) regressor is used for online learning. The script evaluates and compares the performance of these models, providing regression scores for both approaches.

This additional analysis provides insights into the historical trends of weather variables and enables a comparison between batch and online learning models. It offers a holistic view of the script's capabilities, combining real-time streaming and historical analysis for a comprehensive understanding of weather data.





4 Combined Impact

The extended script showcases a versatile and powerful tool for real-time weather data analysis and prediction. By combining online machine learning with real-time data streaming and historical data analysis, the script provides a comprehensive solution for monitoring, analyzing, and predicting weather patterns.

The dynamic map visualization allows users to interactively observe real-time weather conditions across different regions, facilitating quick insights into temperature variations. Simultaneously, the historical data analysis provides a deeper understanding of weather patterns over time.

The incorporation of online machine learning with the `river` library ensures adaptability to changing data patterns, making the script suitable for real-time applications. The comparison between batch and online learning models in the historical data analysis offers valuable insights into the trade-offs between these approaches.

In conclusion, the extended script serves as a robust and flexible tool for a wide range of real-time data analysis and prediction tasks. Its modular design allows for easy extension and adaptation to different scenarios, making it a valuable asset in dynamic and evolving environments.

5 Conclusion & Future Directions

The successful construction of the Streamlit dashboard, integration with Kafka, and exploration of temperature trends using the `SGDRegressor` from the `scikit-learn` library

demonstrate the project's effectiveness in providing a comprehensive solution for real-time weather data analysis. The project offers valuable insights into the advantages and limitations of using online machine learning models for real-time applications, paving the way for future improvements and optimizations.

Looking ahead, there are several promising directions for further enhancing the project. One potential area of expansion is the integration of additional APIs from WeatherAPI.com for more comprehensive weather data analysis. The WeatherAPI.com provides a variety of APIs that can be used to fetch different types of weather data, including:

- Current weather data (`/current.json` or `/current.xml`)
- Forecast data (`/forecast.json` or `/forecast.xml`)
- Search or Autocomplete data (`/search.json` or `/search.xml`)
- History data (`/history.json` or `/history.xml`)
- Marine data (`/marine.json` or `/marine.xml`)
- Future data (`/future.json` or `/future.xml`)
- Time Zone data (`/timezone.json` or `/timezone.xml`)
- Sports data (`/sports.json` or `/sports.xml`)
- Astronomy data (`/astronomy.json` or `/astronomy.xml`)
- IP Lookup data (`/ip.json` or `/ip.xml`)

By integrating these APIs into the project, it would be possible to fetch and analyze a wider range of weather data, thereby enhancing the project's capabilities and providing more comprehensive insights into weather patterns and trends. This would also open up new possibilities for machine learning-based weather prediction, potentially leading to more accurate and reliable predictions.

Demo

A live demonstration of the project shows the successful compilation and execution of the code. The Streamlit dashboard displays real-time weather data, and the Kafka integration allows for efficient data streaming.

References

1. WeatherAPI.com. Real-time weather data API.
2. Apache Kafka. A distributed streaming system.
3. Streamlit. The fastest way to build data apps.
4. Scikit-learn. Machine learning in Python.

Contributors

- Vianney Grimaud (@userVG3854)
- Alpha Oumar Barry (@Barry-07)