Fashionable Weather Alerts via Kafka

Team Members: Via Lin, Mikaela McLean, Angel Mary Oviya, Emily Wong, Sharang Saxena

1 | Introduction

As international students new to San Francisco, our team quickly realized that adjusting to the city's rapidly changing weather posed a daily challenge. San Francisco's Mediterranean-style climate means daytime temperatures range from approximately 9 °C to 19 °C during spring, reaching only up to 23 °C in late summer *(Weather Spark, 2025)*. Nighttime temperatures commonly drop to around 8 °C *(U.S. Climate Data, 2025)*, and persistent coastal winds often exceeding 22 kph during spring months *(World Weather Online, 2025)*—make the evenings feel significantly colder. These rapid shifts created a practical question for us: what jacket should we wear when class starts warm but ends in cold, windy darkness?

To address this, our project leveraged Apache Kafka to build a real-time weather recommendation system based on live data feeds. We designed a Kafka producer that collects weather conditions in San Francisco and a Kafka consumer that interprets this data into actionable clothing advice.

2 | Kafka Producer Function Overview

For the producer, we used the Weatherstack API to fetch live San Francisco weather data. Our Python script was configured to pull current temperature, wind speed, and precipitation values at 1 minute intervals. Each fetched weather record was passed through a rule-based classification system that we developed. In our classification logic, any measurable precipitation triggered a "Raincoat" recommendation. For temperature-only cases, we mapped 0 °C or below to a "Heavy Jacket," 1–10 °C to a "Medium Jacket," 11–18 °C to a "Light Jacket," and above 18 °C to "No Jacket." We further enhanced our model to account for high wind speeds; if wind speeds exceeded 25 kph, we escalated the recommendation by one level to reflect the impact of wind chill on perceived temperature.

After classification, we packaged the timestamp, location, weather metrics, and jacket recommendation into a JSON object. Using the KafkaProducer class from the kafka-python library, we serialized the data and published it to the weather topic on our local Kafka server. *The producer output can be found in section 8 of our report.*

3 | Kafka Consumer Overview

On the receiving end, our consumer script was responsible for subscribing to the weather topic and continuously listening for new weather updates. It used the KafkaConsumer class with a custom JSON deserializer to correctly interpret the incoming messages.

Each record's timestamp was converted into a human-readable format, and the location, temperature, wind speed, precipitation, and recommended jacket type were neatly formatted into an easily readable line. This allowed real-time monitoring of weather changes and clothing recommendations through the terminal, providing immediate value to students trying to make more weather friendly clothing choices. *The consumer output can be found in section 9 of our report.*

4 | Troubleshooting the JSON Serialization Issue

During initial development, our team encountered a major obstacle: although the producer was successfully sending messages to Kafka, the consumer could not parse them properly. After debugging, we realized that the producer was outputting data in raw Python dictionary format rather than JSON, causing deserialization errors in the consumer. We resolved this by explicitly setting a JSON serializer when initializing the Kafka producer. However, because Kafka retained the earlier malformed messages, our consumer kept failing even after fixing the producer code. To solve this, we had to restart the topic by deleting the old one and recreating it from scratch. This experience highlighted for us the persistence of Kafka's message storage and the importance of ensuring consistent message formats from the start.

5 | Real-World Application: "SF Jacket Alert"

While we did not develop a standalone application, our Kafka pipeline provides the foundation for a feature we envision as "SF Jacket Alert", a real-time recommendation engine that could easily be integrated into weather apps already used by most students. Rather than presenting raw data like temperature, wind speed, and precipitation, which still requires interpretation, this system would offer simple, direct advice: "Wear a Medium Jacket" or "Bring a Raincoat."

By embedding our Kafka-driven recommendations into a standard weather app interface, the system would convert environmental facts into actionable insights, removing ambiguity and easing cognitive load. Instead of asking users to interpret whether 14 °C with 26 kph winds warrants a jacket, the app would simply tell them what to wear. This approach eliminates guesswork, particularly for international students unfamiliar with local weather patterns, and delivers value by translating technical data into immediate, real-world decisions.

Behind the scenes, our Kafka producer would continue to fetch updated weather metrics from the Weatherstack API and push structured messages to the Kafka topic. A consumer service could then evaluate the conditions and update the interface in real time, enabling a seamless, intelligent layer atop the traditional weather feed. This concept highlights how streaming infrastructure can directly improve user experience — not by adding more data, but by making data more useful.

6 | Future Scalability and Extensions

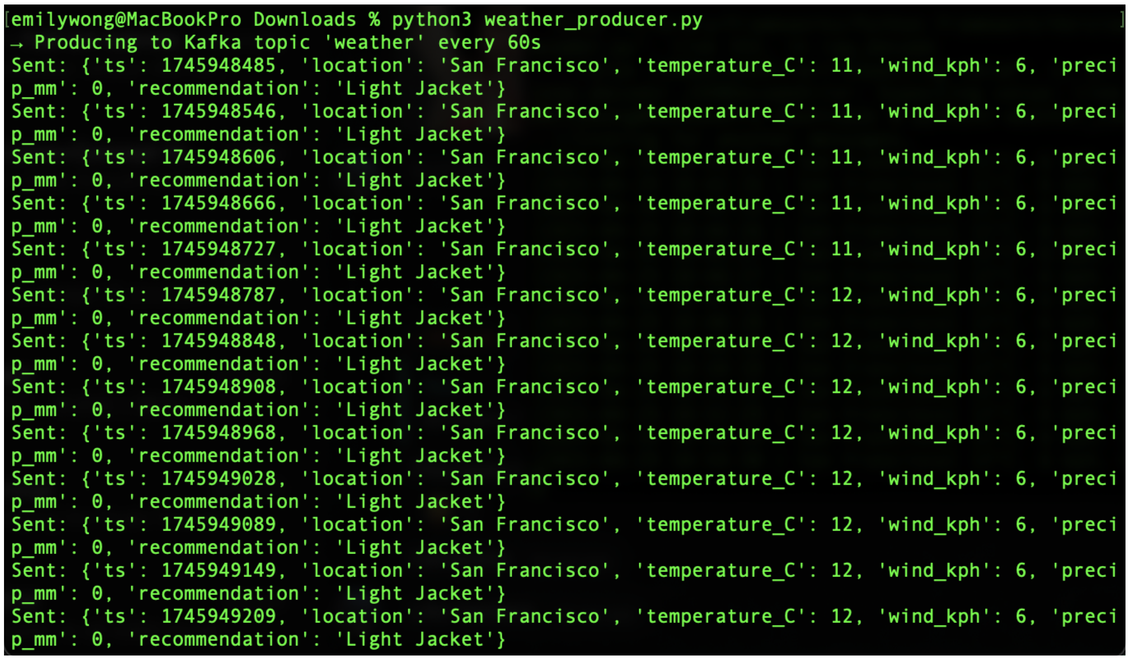
Although our current setup uses a straightforward rule-based classification system, we designed it with future extensibility in mind. One direction for future work would be integrating a machine learning model that could offer more personalized jacket recommendations. These models could take into account additional variables like humidity, UV index, or even a student's own comfort preferences and historical feedback.

Kafka's architecture supports this kind of evolution naturally. Because producers and consumers are decoupled, we could improve our logic or add new consumer features without changing how data is produced. This allows for parallel development and experimentation, making the system adaptable to new requirements or expanded use cases. While these enhancements were outside the scope of our current work, they represent promising next steps should this project be developed further.

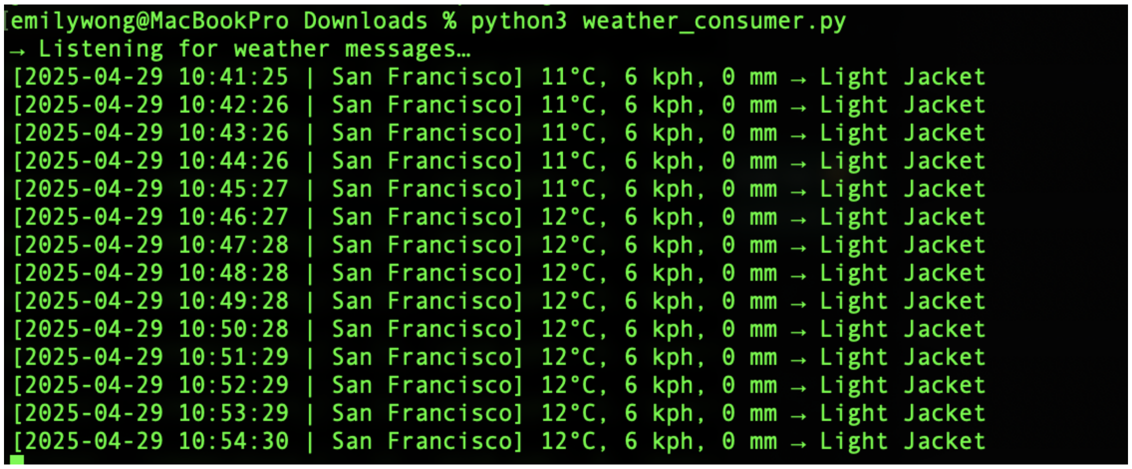
7 | Conclusion

In completing this project, our team gained hands-on experience designing a real-time data pipeline using Apache Kafka, built around a problem we personally face as international students adapting to San Francisco's unpredictable climate. By integrating live weather data from the Weatherstack API, applying context-aware classification logic, and resolving challenges like serialization and stale topic data, we developed a functioning producer-consumer system that transforms raw environmental data into timely jacket recommendations. While our implementation was focused on the core Kafka components, it opens the door to broader applications—from integration with everyday weather apps to the potential for machine learning–driven personalization. Ultimately, this project demonstrated how stream processing can move beyond technical infrastructure to solve practical, human-centered problems.

8 | Producer Output


```
[emilywong@MacBookPro Downloads % python3 weather_producer.py                   ]
→ Producing to Kafka topic 'weather' every 60s
Sent: {'ts': 1745948485, 'location': 'San Francisco', 'temperature_C': 11, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948546, 'location': 'San Francisco', 'temperature_C': 11, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948606, 'location': 'San Francisco', 'temperature_C': 11, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948666, 'location': 'San Francisco', 'temperature_C': 11, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948727, 'location': 'San Francisco', 'temperature_C': 11, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948787, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948848, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948908, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745948968, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745949028, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745949089, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745949149, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
Sent: {'ts': 1745949209, 'location': 'San Francisco', 'temperature_C': 12, 'wind_kph': 6, 'preci
p_mm': 0, 'recommendation': 'Light Jacket'}
```

9 | Consumer Output


```
[emilywong@MacBookPro Downloads % python3 weather_consumer.py
→ Listening for weather messages…
[2025-04-29 10:41:25 | San Francisco] 11°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:42:26 | San Francisco] 11°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:43:26 | San Francisco] 11°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:44:26 | San Francisco] 11°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:45:27 | San Francisco] 11°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:46:27 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:47:28 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:48:28 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:49:28 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:50:28 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:51:29 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:52:29 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:53:29 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
[2025-04-29 10:54:30 | San Francisco] 12°C, 6 kph, 0 mm → Light Jacket
```

10 | Bibliography

1. *Weather Spark. Climate and Average Weather Year Round in San Francisco. Retrieved April 2025, from https://weatherspark.com/y/1550/Average-Weather-in-San-Francisco-California-United-States-Year-Round*
2. *U.S. Climate Data. San Francisco Average Monthly Temperatures and Precipitation. Retrieved April 2025, from https://www.usclimatedata.com/climate/san-francisco/california/united-states/usca0987*
3. *World Weather Online. San Francisco Average Wind Speed. Retrieved April 2025, from https://www.worldweatheronline.com/san-francisco-weather-averages/california/us.aspx*
4. *Weatherstack. Real-Time Weather API Documentation. Retrieved from https://weatherstack.com/documentation*