

Weather and Event Recommendation Pipeline

DSAN5500: Data Structures, Objects, and Algorithms in Python

Samantha Wang

Georgetown University

May 01, 2025

Table of contents

1	Abstract	3
2	Introduction	3
3	Methodology	3
3.1	Data Sources and Extraction	3
3.2	Data Transformation and Validation	4
3.3	Data Storage and Versioning	4
3.4	Frontend Application Development	4
4	Code Snippets and Pipeline Structure	5
5	Results and Website	7
5.1	Website Overview	7
5.2	Weather Dashboard Screenshot	7
5.3	Event Recommendation Screenshot	7
6	Prefect UI Screenshot	8
7	Conclusion	9
8	References	9

1 Abstract

The increasing availability of real-time data sources provides an opportunity to build dynamic applications that enhance everyday decision-making. This project presents the development of an automated ETL (Extract, Transform, Load) pipeline combined with a Streamlit web application that delivers daily 5-day weather forecasts and curated event recommendations for New York City. The pipeline collects weather data from OpenWeatherMap and event listings from Ticketmaster APIs, validates and enriches the data, and automatically updates CSV datasets on GitHub. These datasets feed into a publicly deployed Streamlit app that presents users with actionable recommendations.

The project demonstrates the integration of data engineering workflows, cloud storage synchronization, and interactive web design to address the needs of users seeking weather-aware event planning.

2 Introduction

Weather fluctuations and urban events are deeply intertwined, influencing how people plan their leisure activities. Despite the existence of numerous weather apps and event platforms, users often need to manually check different sources before making decisions. This project seeks to close that gap by automatically merging weather data with event listings and generating actionable daily recommendations.

Unlike static dashboards, the system designed here continuously updates itself without manual intervention. Using Prefect flows and GitHub for version control, the entire data processing cycle is fully automated. This allows the Streamlit application to always serve fresh information to users.

Key goals include:

- Automating data updates daily.
- Recommending events considering weather suitability.
- Hosting an easily accessible frontend for end users.

3 Methodology

3.1 Data Sources and Extraction

Weather data is sourced from the OpenWeatherMap API, capturing both the current day's conditions and 5-day forecasts at 3-hour intervals. Event data is retrieved from Ticketmaster's Discovery API, targeting categories such as Music, Sports, Arts & Theatre, and Festivals. Extraction tasks include API request management, pagination handling, and ensuring time zone consistency, particularly considering daylight saving time adjustments in New York.

Both APIs were integrated using scheduled Prefect tasks, ensuring that data refreshes daily without manual triggers.

3.2 Data Transformation and Validation

The transformation step standardizes and validates the collected data:

- **Weather Data:** Metrics such as temperature, humidity, wind speed, cloudiness, and precipitation chance are extracted. Only midday forecasts (12 PM) are used to reduce volatility.
- **Event Data:** For each event, metadata including name, date, time, venue, ticket price range, and associated media is captured.

Pandera schemas are implemented to enforce strict data type expectations, preventing corrupted or incomplete records from proceeding.

Additionally, a simple rule-based recommendation system evaluates events based on forecasted weather: - Outdoor events are marked “Recommended” if rain probability is low and temperatures are mild. - Indoor events are prioritized during adverse weather conditions.

3.3 Data Storage and Versioning

Post-validation, datasets are saved into the `/output` directory as CSV files. Using a Git-based workflow embedded within the Prefect pipeline, updated outputs are committed and pushed to the GitHub repository daily.

This design guarantees that the Streamlit frontend always reflects the latest available data without relying on manual file uploads.

Challenges addressed include handling Git merge conflicts during rapid development and implementing pull-rebase strategies to maintain clean commit histories.

3.4 Frontend Application Development

The Streamlit app dynamically loads the most recent datasets. Users can:

- Select a specific date within the upcoming 5-day window.
- View detailed weather summaries.
- Browse events by date and recommendation type.
- Access event booking pages via embedded hyperlinks.

Caching is used strategically (`@st.cache_data`) to minimize unnecessary reloads while ensuring daily freshness (`ttl=86400` seconds).

Deployment is done through Streamlit Community Cloud, offering free public hosting with automatic rebuilds on GitHub changes.

4 Code Snippets and Pipeline Structure

The ETL pipeline is modularized into clear steps for better maintenance and scalability:

- **extract()**: Calls APIs to fetch weather forecasts and event information.
- **transform()**: Converts API results into Pandas DataFrames, validates schema with Pandera, and generates event recommendations based on forecasted weather.
- **load()**: Saves validated DataFrames to `output/` directory as CSV files.
- **github__push()**: Automates Git operations to commit updated files and push to GitHub.

```
@task
def extract():
    weather_api_key = os.getenv("WEATHER_API_KEY")
    event_api_key = os.getenv("EVENT_API_KEY")

    if not weather_api_key or not event_api_key:
        raise ValueError("Missing WEATHER_API_KEY or EVENT_API_KEY in environment variable")

    weather_data = fetch_weather_forecast(weather_api_key)
    event_data = fetch_events_forecast_daily(event_api_key)
    return weather_data, event_data

@task
def transform(weather_data: list, event_data: list):
    weather_df = pd.DataFrame(weather_data)
    event_df = pd.DataFrame(event_data)
    weather_df["date"] = pd.to_datetime(weather_df["date"])
    event_df["event_date"] = pd.to_datetime(event_df["event_date"])

    float_columns = [
        "temperature_celsius", "feels_like", "temp_min", "temp_max",
        "humidity", "pressure", "wind_speed", "cloudiness", "precipitation_chance"
    ]
    weather_df[float_columns] = weather_df[float_columns].astype(float)
    weather_lookup = weather_df.set_index(weather_df["date"].dt.date)

    recommendations = []
    for _, row in event_df.iterrows():
        event_day = row["event_date"].date()
        today_weather = weather_lookup.loc[event_day] if event_day in weather_lookup.index
        if isinstance(today_weather, pd.DataFrame):
            today_weather = today_weather.dropna(subset=["weather_main"]).iloc[0] if not t
        elif today_weather is not None and pd.isna(today_weather.get("weather_main", None)):
            today_weather = None
```

```

        if today_weather is not None:
            rec = generate_recommendation(
                today_weather["temperature_celsius"],
                today_weather["feels_like"],
                today_weather["humidity"],
                today_weather["wind_speed"],
                today_weather["weather_main"],
                today_weather["precipitation_chance"],
                row["venue"]
            )
        else:
            rec = "No Recommendation"

        recommendations.append(rec)

    event_df["recommendation"] = recommendations
    weather_df = validate_weather(weather_df)
    event_df = validate_events(event_df)
    return weather_df.reset_index(drop=True), event_df.reset_index(drop=True)

@task
def load(weather_df: pd.DataFrame, event_df: pd.DataFrame):
    save_to_csv(weather_df, event_df)

@task
def github_push():
    upload_to_github("output/weather_forecast.csv", "samantha0820/weather-event-etl", "output")
    upload_to_github("output/events_forecast.csv", "samantha0820/weather-event-etl", "output")

@flow(name="Daily ETL Pipeline")
def etl_pipeline():
    weather_data, event_data = extract()
    weather_df, event_df = transform(weather_data, event_data)
    load(weather_df, event_df)
    github_push()

```

Each task isolates concerns: API fetching, transformation and validation, storage, and synchronization.

5 Results and Website

5.1 Website Overview

The Streamlit app provides:

- Weather metrics for the next 5 days.
- Event cards dynamically filtered by date and recommendation level.

5.2 Weather Dashboard Screenshot

The weather dashboard provides an at-a-glance view of key forecast indicators, helping users quickly assess temperature, humidity, precipitation chance, and general conditions. User interactions are streamlined with dropdown selectors and metric displays.

5-Day Weather & Event Recommendations

Weather Summary

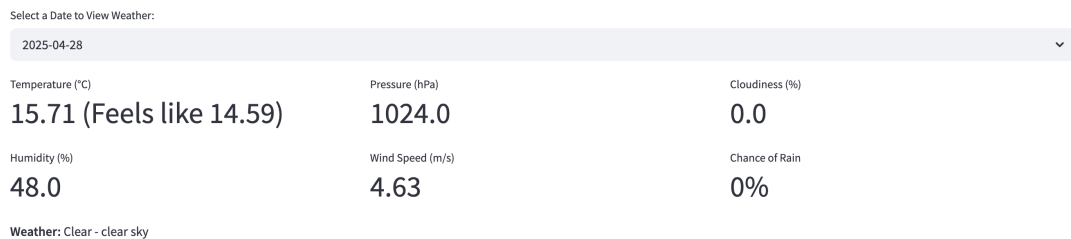


Figure 1: Weather Dashboard Screenshot

5.3 Event Recommendation Screenshot

The event recommendation section surfaces concerts, festivals, and other activities, pre-filtered by the weather's favorability.

Events are presented as rich, card-style displays with images, times, venues, ticket prices, and clickable links.

5-Day Events


Select a Date for Events:

2025-04-28

Filter by Recommendation:

All

17 Events on April 28, 2025



[Vince Giordano and the Nighthawks](#)

Venue: Birdland Theater, New York

Time: 17:30:00

Price: 0.00—0.00

Recommendation: Recommended (Outdoor)

Figure 2: Event Recommendation Screenshot

Live Site: [Streamlit App](#)

Source Code: [GitHub Repository](#)

6 Prefect UI Screenshot

Below is the Prefect Cloud UI monitoring the daily ETL flow.

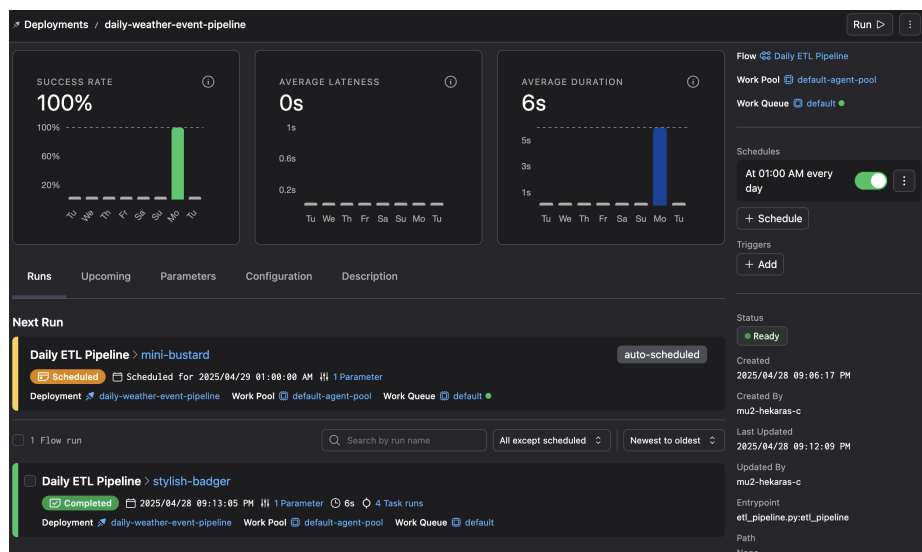


Figure 3: Prefect UI Screenshot

The ETL pipeline has been consistently completing runs daily with automatic CSV artifact uploads.

7 Conclusion

This project demonstrates how a small-scale, fully automated data pipeline coupled with a responsive web application can address a real-world need: aligning event discovery with forecasted weather conditions. By seamlessly integrating weather data and event information, users are empowered to make more informed, practical decisions regarding their daily leisure activities.

The automation of both backend processes (data extraction, validation, and storage) and frontend application updates ensures that the system remains highly scalable, sustainable, and low-maintenance. The ETL pipeline is scheduled to refresh data daily without manual intervention, providing users with the latest recommendations based on current forecasts. Similarly, the Streamlit-based frontend automatically reads the updated datasets and dynamically presents new insights.

The modular design of the framework also enables easy generalization to other cities or event platforms, suggesting promising avenues for further development. Future work could expand the project by incorporating richer event metadata, integrating multiple weather APIs for greater robustness, or adding user personalization features such as location-aware filtering.

Ultimately, by combining data reliability, interface usability, and continuous daily updates, the application delivers tangible practical value. It highlights the power of lightweight data engineering pipelines and user-centered web interfaces to enhance everyday decision-making, showcasing a practical model that can be scaled or adapted across diverse contexts.

8 References

- OpenWeatherMap API. <https://openweathermap.org/api>
- Ticketmaster Discovery API. <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>
- Streamlit Documentation. <https://docs.streamlit.io/>
- Pandera Library. <https://pandera.readthedocs.io/>
- Prefect Documentation. <https://docs.prefect.io/>