# Technical Report: Final Project DS 5110: Weather Data Integration Pipeline and Dashboard


Rohit Sunilkumar Hooda
hooda.r@northeastern.edu
Rimsha Kayastha
kayastha.r@northeastern.edu
Tri Watanasuparp
watanasuparp.t@northeastern.edu
Khoury College of Computer Sciences


December 9, 2024

## Contents

# 1 Introduction

Many people start their day by checking the weather forecast to plan activities such as commuting, exercising, or scheduling events. Given how essential accurate weather information is to daily life, it's important to have reliable and comprehensive tools at our disposal. This project seeks to fill that need by integrating data from multiple sources and offering a detailed weather dashboard that provides both current forecasts, for future preparations, and historical data, to be aware of the ongoing weather trends. Additionally, it incorporates interactive, user-friendly visualizations that empower users to analyze trends, draw relevant conclusions, and make informed decisions based on weather patterns. This combination of real-time updates and historical context aims to enhance users' ability to anticipate weather conditions and better plan their activities. This project focuses on Massachusetts, a US state, in particular for work on a more detailed, smaller scale.

# 2 Literature Review

The Arcgis weather dashboard (`https://www.arcgis.com/apps/dashboards/f7f008a8452747d0a8d86ed69d77cdc4`) is a powerful tool designed to display real-time weather data with an emphasis on USA weather alerts, including storms, heat waves, and hurricane watches. Central to the dashboard is an interactive map, as you can see in Figure 1, that users can easily navigate—dragging, zooming in, or zooming out—to access weather information specific to any location. This interactivity provides an accessible way for users to obtain localized data, making it especially valuable for monitoring conditions and staying informed about potential hazards in targeted areas.


Figure 1: Arcgis weather dashboard

## 2.1 Ambient Weather Dashboard

The Ambient Weather dashboard (`https://ambientweather.net/`) provides detailed, real-time weather data sourced from personal weather stations, offering hyper-localized information that is essential for users who need precise, neighborhood-specific updates. As shown in Figure 2, it tracks key weather metrics such as temperature, humidity, wind speed, and rainfall, ensuring a comprehensive view of current conditions. Furthermore, the dashboard is highly customizable, allowing users to choose the specific weather parameters they wish to monitor and arrange the data widgets to suit their needs, whether for gardening, event planning, or professional purposes. Additionally, it includes historical data analysis capabilities, enabling users to observe weather trends over time. This tailored and interactive approach makes the Ambient Weather dashboard a valuable resource for anyone wanting to stay informed about local weather conditions.

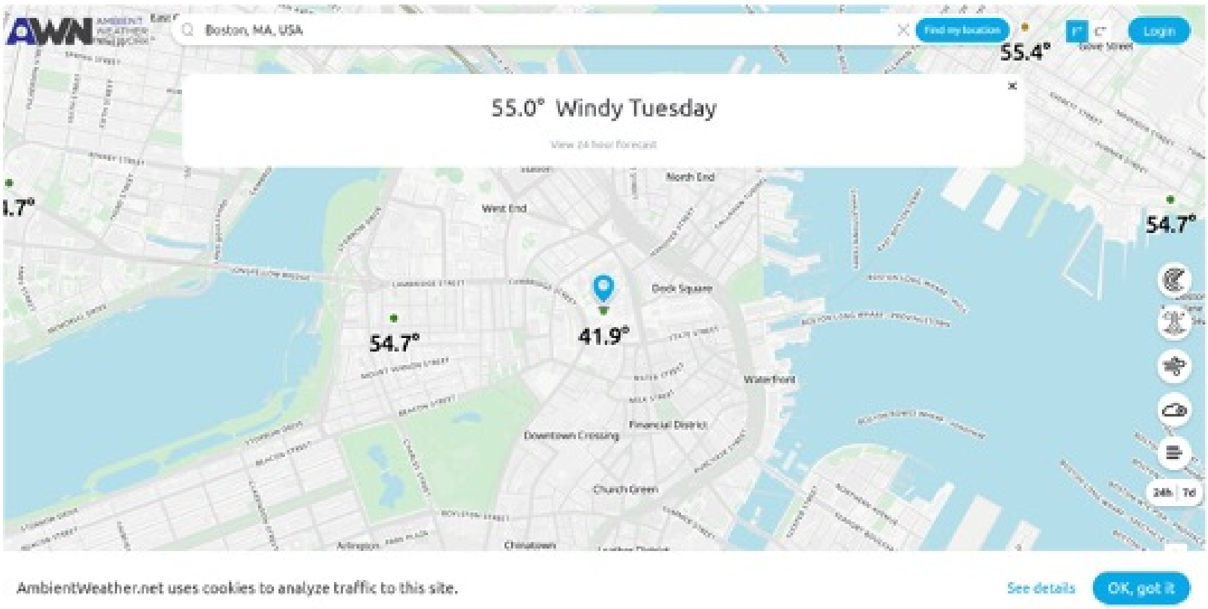


Figure 2: Ambient Weather dashboard

# 3 Project Structure and Methodology

As shown in Figure 3, the flow of this project has three sections: Data sources, transformation, and deliverables. The first section involves retrieving data from multiple sources and integrating them into a singular pipeline, feeding them into the next section of data processing and transformation. Data transformation refers to cleaning the data to address null values and outliers, and pre-processing refers to standardizing the data into a format that makes it more readable for users. Finally, the last section of the project is to produce readable visualizations that help draw conclusions and portray them in distributable formats.

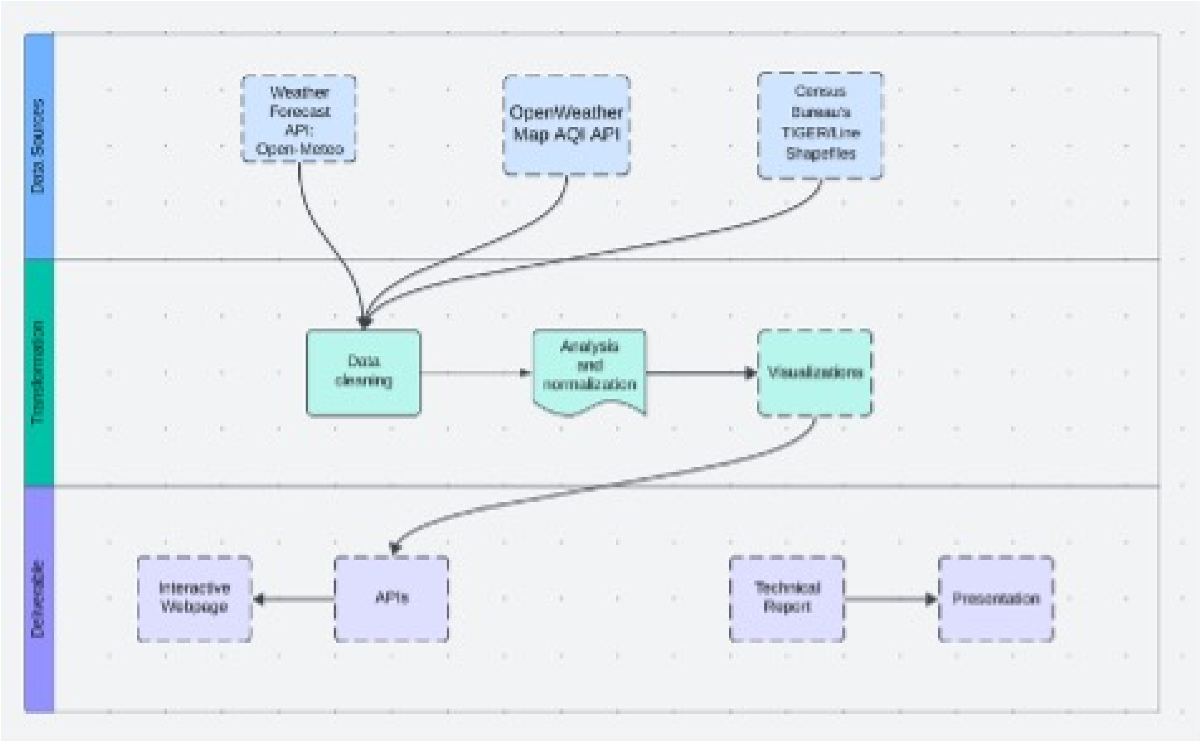


Figure 3: Project Process

## 3.1 Data Pre-processing and Retrieval

Our weather data is retrieved from three sources through API calls: Mateo, AQI, and Census Bureau's TIGER/Line Shapefiles. This data is pipelined into a single data frame, which is then cleaned, analyzed, and plotted for inference. The raw, combined data includes the following features: latitude, longitude, weather code, daylight duration, sunshine duration, max temperature, min temperature, max UV index reached, max precipitation probability, and AQI value. These features are then filtered down to the 12 features we are focusing on for the dashboard: date, county, latitude, longitude, weather code, daylight duration, sunshine duration, max temperature, min temperature, max UV index, max precipitation probability, and AQI value.

## 3.2 Data Analysis

Gathering data from the three sources resulted in obtaining more than 25 features, which were funneled down into the 12 relevant features mentioned in the previous section. Consequently, analysis was performed on the spread of data for those features. For analysis, the features were divided into three groups based on their data scale.

Figure 4 shows the bar chart for the spread of data for sunshine and daylight duration, where each unit represents two hours.

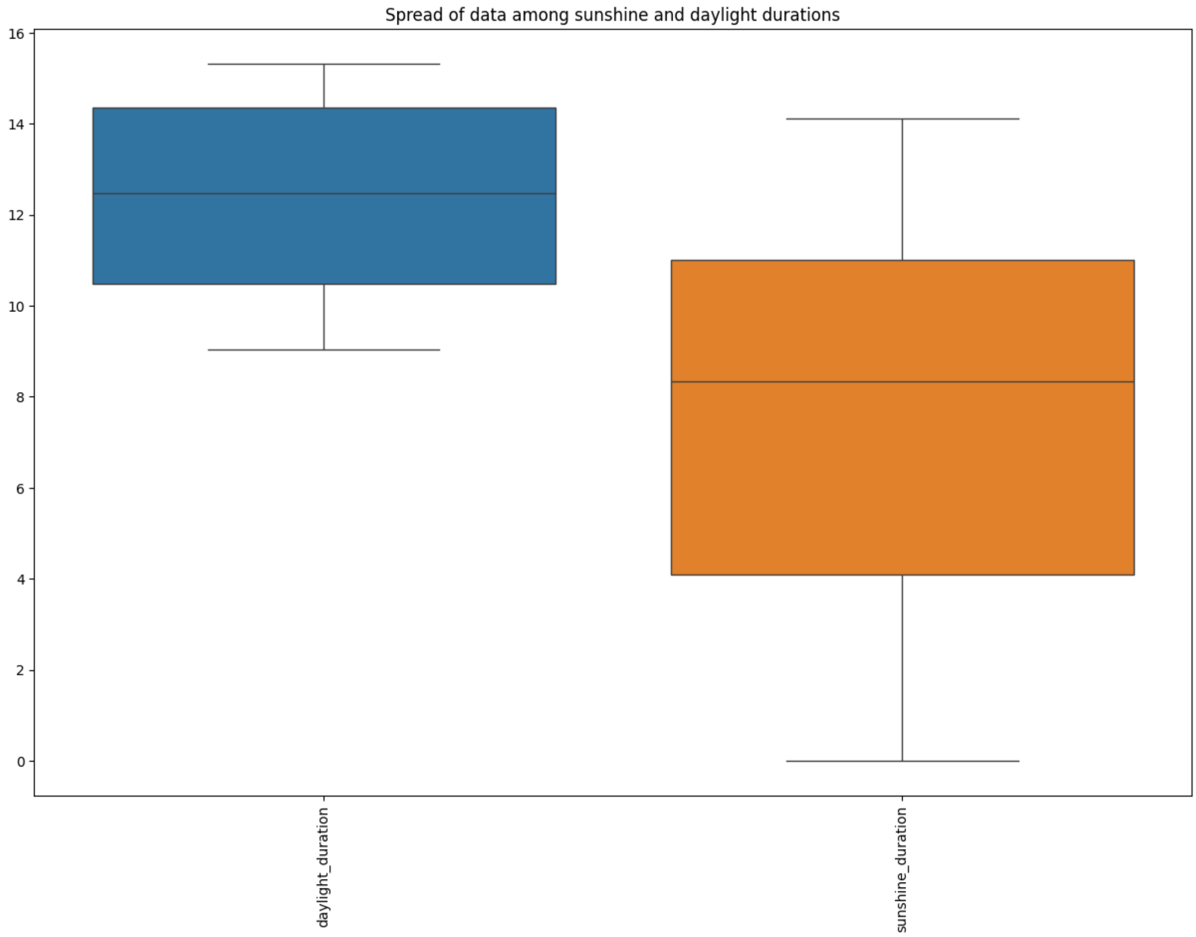


Figure 4: Sunshine-Daylight Duration Data Spread

Figure 5 shows the spread of minimum and maximum temperature data, and Figure 6 shows the spread of the remaining features.

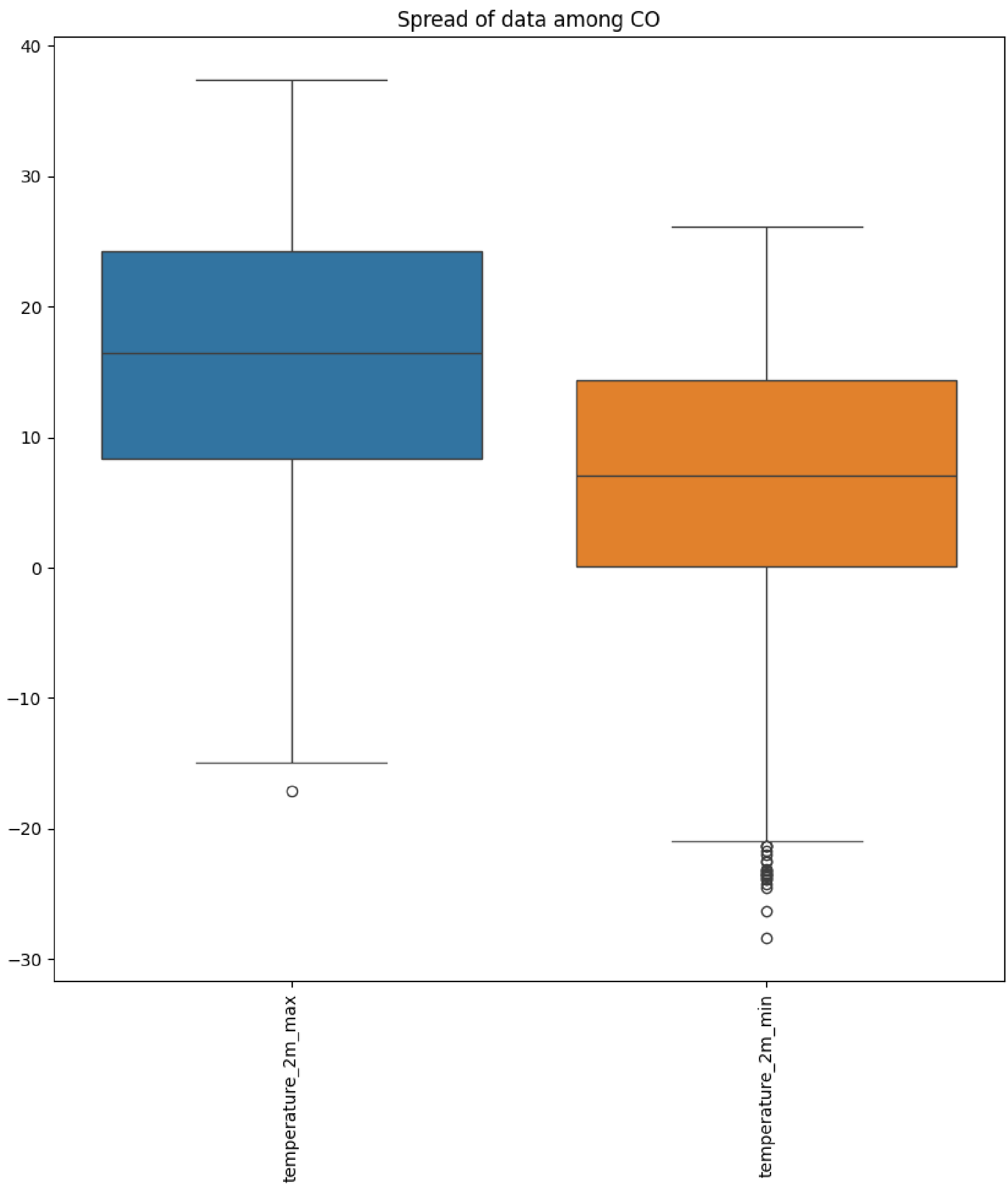


Figure 5: Min and Max Temperature Data Spread

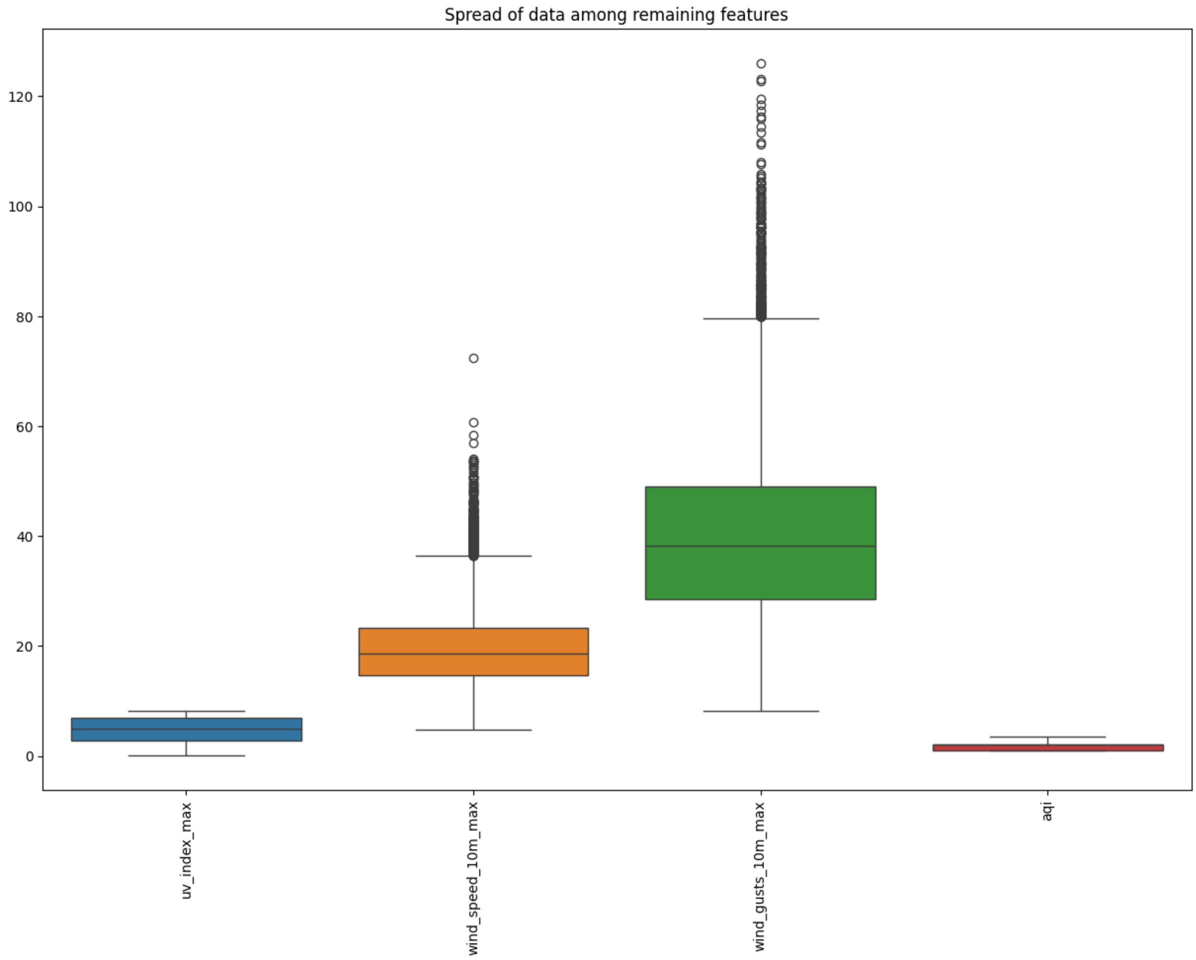


Figure 6: Data Spread Among the Remaining Features

### 3.3 Data Cleaning

The bar charts in the previous section reveal many outliers that need adjustment. Using the quartiles from the data spread, the data points are clipped to an upper and lower bound, such that the lower bound is calculated as $Q_1 - 1.5 \times \text{IQR}$ and the upper bound as $Q_3 + 1.5 \times \text{IQR}$, where $Q_1$ is the first quartile, $Q_3$ is the third quartile, and IQR is the interquartile range.

## 4 Webapp Architecture

Present the results of the analysis. Use tables, figures, and charts to support the findings.

### 4.1 Front End Architecture

The frontend is written in HTML, CSS, JavaScript and we are using Axios for API communication and AnyChart library for data visualization. The frontend will enable users to select weather metrics, date ranges, and visualize data interactively.

### 4.1.1 Visualizing the Weather Data on the Frontend

The `fetchWeatherData()` function serves as a bridge between the frontend and backend, handling user input, data processing, and visualization. Here's how the frontend workflow is structured:

1. **User Input Retrieval**
   - The function gathers inputs from the DOM, including the selected county, metrics (e.g., temperature, precipitation), and date range.
2. **Constructing API Request**
   - A GET request is dynamically built for the `/weather` endpoint, incorporating user-selected parameters.
3. **Processing the API Response**
   - The JSON response from the backend is parsed to extract:
     - `info_type`: Represents the name of the weather metric (e.g., temperature or rainfall).
     - `values`: An array containing date-value pairs, formatted for charting.
4. **Rendering with AnyChart**
   - The extracted data is fed into the AnyChart library to generate a line chart. Key aspects include:
     - Each metric is visualized as a distinct series, represented with a unique color for clarity and easy differentiation.

This setup ensures a seamless user experience by combining real-time interactivity with visually engaging data representation.

## 4.2 Backend Architecture

The backend API for the Weather Visualization App is developed using Flask, ensuring a lightweight yet robust architecture for handling requests. For data extraction, we utilized multiple sources such as OpenMateo, OpenWeatherAPI, and TIGER Shapefiles, enabling a rich and comprehensive dataset. To prepare the data for analysis and visualization, we employed Python libraries like Pandas for data cleaning and manipulation. Seaborn was leveraged for creating informative visualizations, and Folium was used to generate detailed maps of Massachusetts counties. This combination of tools ensures efficient data processing and visually appealing representations, creating a seamless backend experience.

### 4.2.1 Routes

1. **Index Route: (GET /)**
   - Purpose: Displays an interactive map of Massachusetts with county-level weather data.

- Implementation:
  - Uses Folium to create a map.
  - Dynamically generates popups for each county with real-time weather data.
  - Fetches weather data using the `county_weather_data` dictionary.
  - Encodes county boundaries into GeoJSON to render shapes on the map.

2. **Weather Data Route (GET /weather)**
   - Purpose: Fetches historical weather data for a specific county and time range.
   - Parameters:
     - `countyName`: Name of the county.
     - `typeOfInformation`: List of weather metrics (e.g., temperature, humidity).
     - `fromDate, toDate`: Time range for the data query.
   - Implementation:
     - Filters the historical weather dataset stored in `data_frame` using the provided parameters.
     - Returns a structured JSON response with selected metrics and their corresponding values over the specified time period.
   - Response:
     - Success: JSON object containing filtered weather data for the specified county and date range.
     - Error: Returns 400: Missing required parameters or Returns 404: No data available for the given criteria.

**Example Response:**

```
{
  "county_name": "Middlesex",
  "data": [
    {
      "info_type": "temperature_2m_max",
      "values": [{"Date": "2024-01-01", "value": 45.3}, ...]
    },
    {
      "info_type": "precipitation",
      "values": [{"Date": "2024-01-01", "value": 0.2}, ...]
    }
  ]
}
```

# 5 Dashboard Design and Visualizations

Interpret the results and discuss their implications. Compare the findings with the literature review and explain any discrepancies.

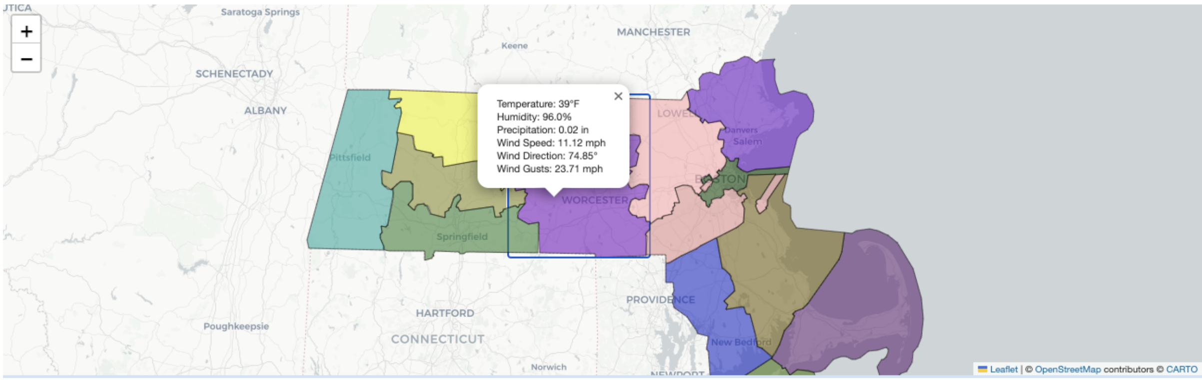


Figure 7: Choropleth Map - Massachusetts

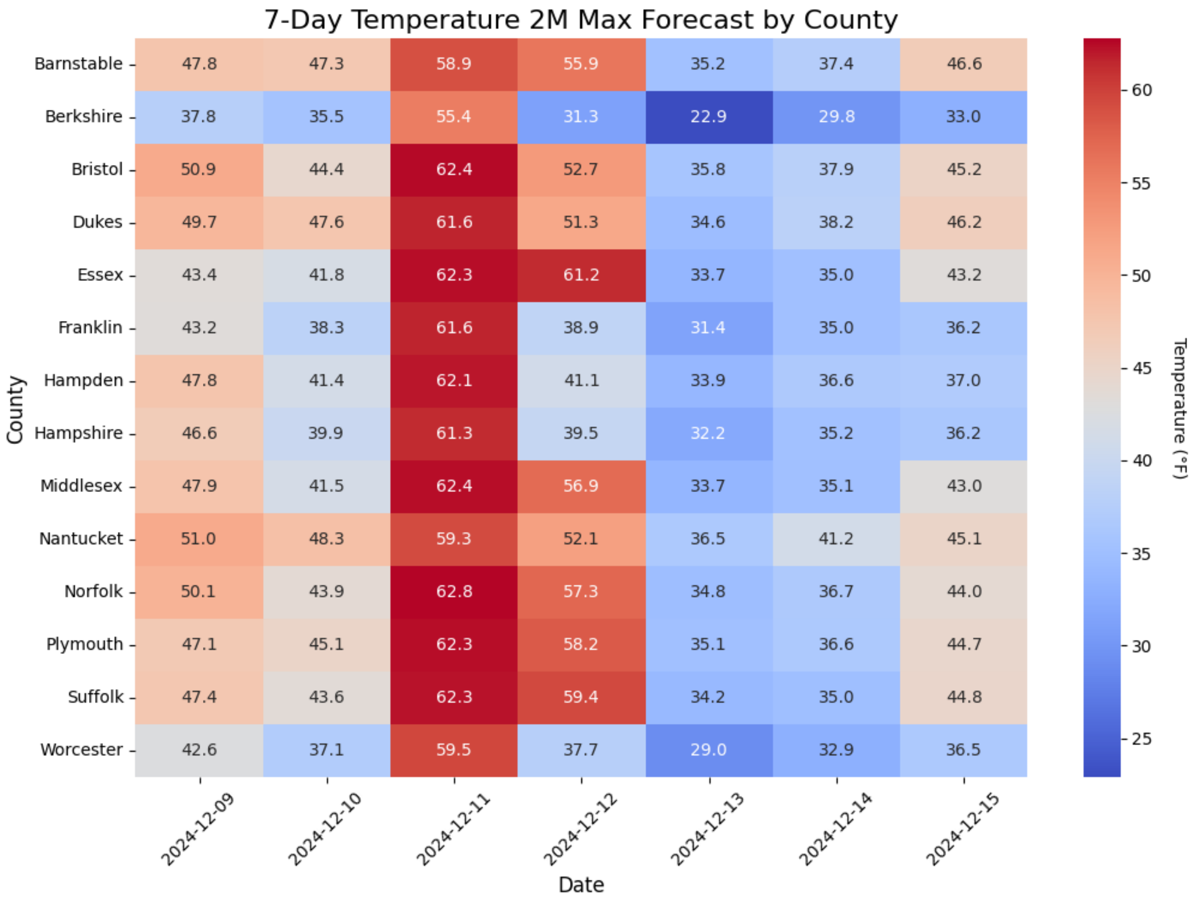


Figure 8: 7-day Temperature Forecast Heatmap

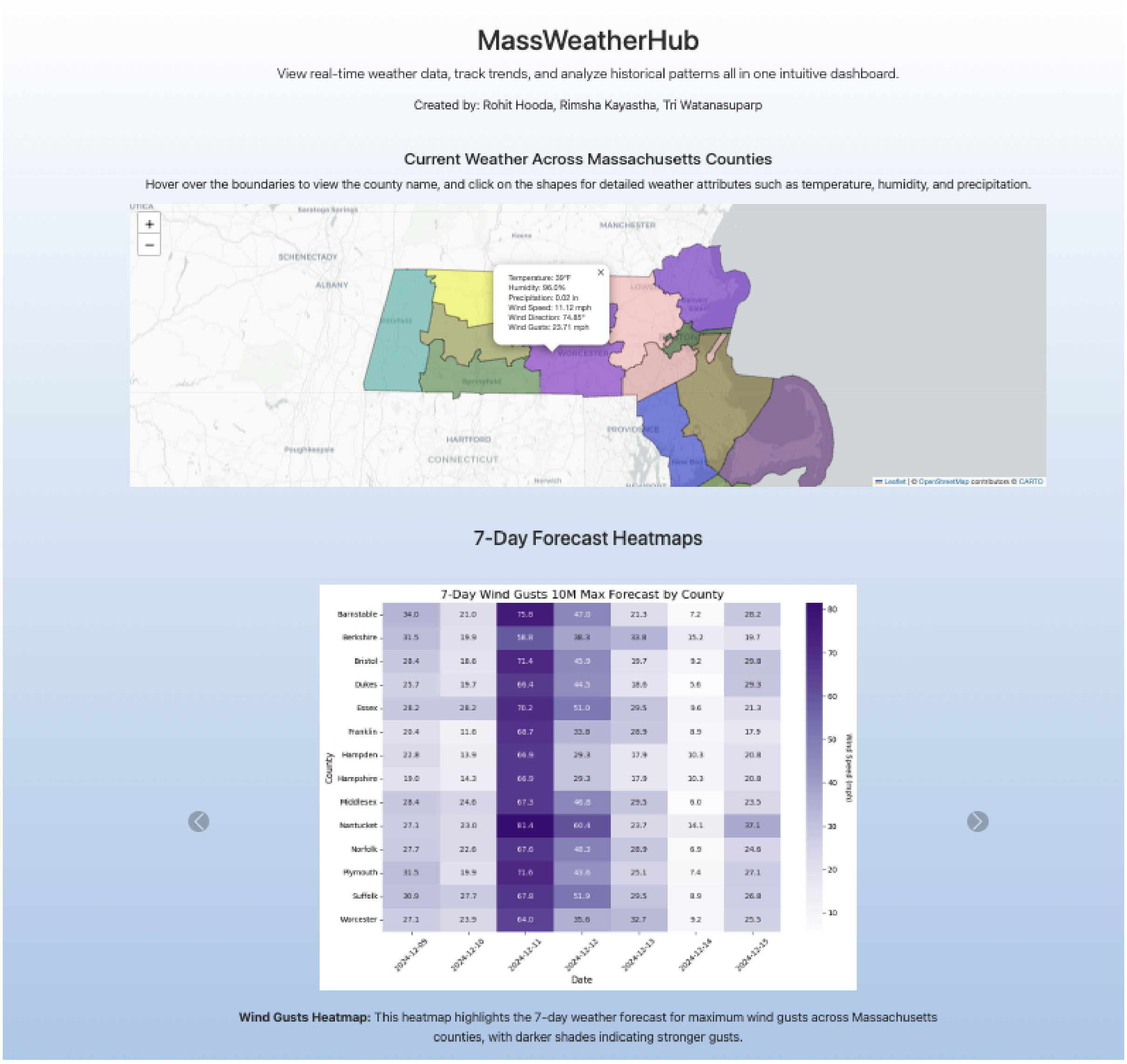


Figure 9: Website Dashboard

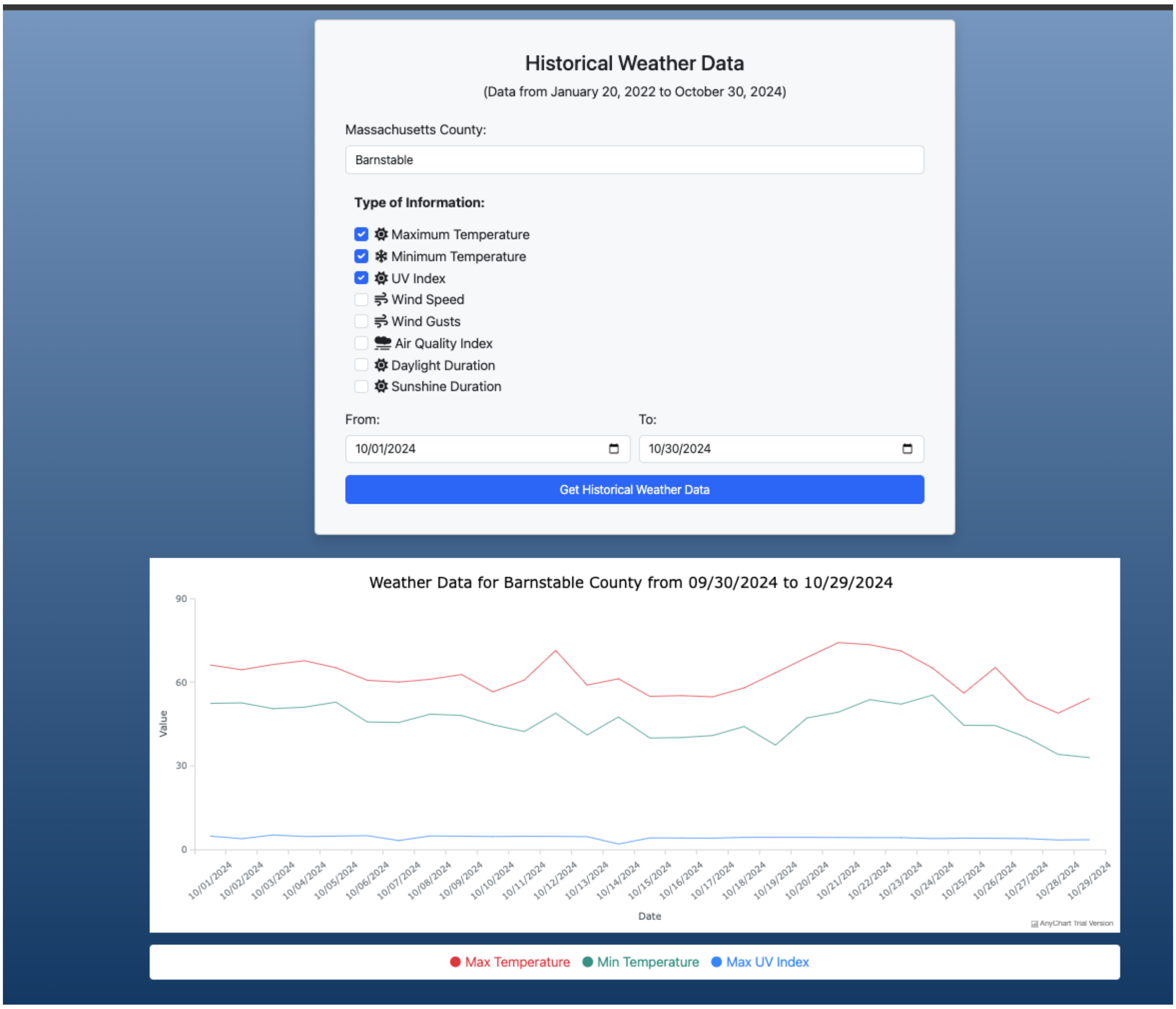


Figure 10: Historical Data Plot

# 6 Conclusion

In this project we successfully developed a weather dashboard with interactive visualizations for real-time and historical weather data. The system has been built to provide users with an easy-to-use interface to view and analyze weather data. However, there are opportunities for expanding the project further.

## 6.1 Future Work

Future work for the weather app could include the following enhancements:

- Expand weather app coverage to all U.S. states.
- Integrate additional data sources and features.
- Enable data retrieval via the user's current location.
- Host the website on the cloud.
- Integrate email subscription for weather alerts.

# 7 References


1. Open-Meteo API

2. MassGIS shapefiles

3. NOAA (2023). Weather Data API Documentation.

4. OpenWeatherMap (2024). Historical Data Overview.

5. https://www.arcgis.com/apps/dashboards/f7f008a8452747d0a8d86ed69d77cdc4

6. https://ambientweather.net/


# A Appendix A: Code

## A.1 Data Analysis Code

Below is the code used for data analysis and cleaning:

```
massachusettsCounties = [
    "Barnstable",
    "Berkshire",
    "Bristol",
    "Dukes",
    "Essex",
    "Franklin",
    "Hampden",
    "Hampshire",
    "Middlesex",
    "Nantucket",
    "Norfolk",
    "Plymouth",
    "Suffolk",
    "Worcester"
]

"""Retrieving data from Mateo"""

import pandas as pd

Mateo_df1 = pd.read_csv('../Dataset/raw_data/2022
    _massachusetts_counties_weather_data.csv')
Mateo_df2 = pd.read_csv('../Dataset/raw_data/2023
    _massachusetts_counties_weather_data.csv')
Mateo_df3 = pd.read_csv('../Dataset/raw_data/2024
    _massachusetts_counties_weather_data.csv')
Mateo_df = pd.concat([Mateo_df1, Mateo_df2, Mateo_df3])

raw_df = Mateo_df.copy()
for county in massachusettsCounties:
    AQI_df = pd.read_csv(f'../Dataset/raw_data/AQI/{county}
    _air_pollution_history.csv')
    # check if columns already exist
    if 'aqi' in raw_df.columns:
```

```python
        raw_df.update(AQI_df.set_index('dt'))
    else:
        raw_df = pd.merge(raw_df, AQI_df, left_on='date', right_on='dt', how='left')

raw_df.head()

"""Data Cleaning"""

filter_columns = ['date', 'county', 'latitude', 'longitude',
                  'temperature_2m_max', 'temperature_2m_min',
                  'uv_index_max', 'wind_speed_10m_max', 'wind_gusts_10m_max','aqi', 'daylight_duration', 'sunshine_duration', 'weather_code']
raw_df_filtered = raw_df[filter_columns]
raw_df_filtered

raw_df_summary = raw_df_filtered.describe()
raw_df_summary

# check for null values
raw_df_filtered.isnull().sum()

# Outliers

min_values = {
    'uv_index_max': 0.1, 'aqi': 1.0
    }

for col in min_values.keys():
    min = raw_df_summary.loc['min', col]
    q1 = raw_df_summary.loc['25%', col]
    q3 = raw_df_summary.loc['75%', col]
    max = raw_df_summary.loc['max', col]
    IQR = q3-q1

    for i in range(len(raw_df_filtered)):
        val =  raw_df_filtered[col][i]
        if (val<(q1-1.5*IQR )):
            raw_df_filtered.loc[i, col] = min_values[col]
        elif (val>(q3+1.5*IQR )):
            raw_df_filtered.loc[i, col] = (q3+1.5*IQR )

raw_df_filtered.describe()

# Create the weather code mapping dictionary
weather_code_map = {
    0: 'Clear sky',
    1: 'Mainly clear, partly cloudy, and overcast',
    2: 'Mainly clear, partly cloudy, and overcast',
    3: 'Mainly clear, partly cloudy, and overcast',
    45: 'Fog and depositing rime fog',
    48: 'Fog and depositing rime fog',
    51: 'Drizzle: Light, moderate, and dense intensity',
    53: 'Drizzle: Light, moderate, and dense intensity',
    55: 'Drizzle: Light, moderate, and dense intensity',
    56: 'Freezing Drizzle: Light and dense intensity',
    57: 'Freezing Drizzle: Light and dense intensity',
```

```
    61: 'Rain: Slight, moderate and heavy intensity',
    63: 'Rain: Slight, moderate and heavy intensity',
    65: 'Rain: Slight, moderate and heavy intensity',
    66: 'Freezing Rain: Light and heavy intensity',
    67: 'Freezing Rain: Light and heavy intensity',
    71: 'Snow fall: Slight, moderate, and heavy intensity',
    73: 'Snow fall: Slight, moderate, and heavy intensity',
    75: 'Snow fall: Slight, moderate, and heavy intensity',
    77: 'Snow grains',
    80: 'Rain showers: Slight, moderate, and violent',
    81: 'Rain showers: Slight, moderate, and violent',
    82: 'Rain showers: Slight, moderate, and violent',
    85: 'Snow showers slight and heavy',
    86: 'Snow showers slight and heavy',
    95: 'Thunderstorm: Slight or moderate',
    96: 'Thunderstorm with slight and heavy hail',
    99: 'Thunderstorm with slight and heavy hail'
}

raw_df_filtered['weather_code'] = raw_df_filtered['weather_code'].map(
    weather_code_map)
raw_df_filtered.head()

# Forward fill missing values for AQI (use the last valid value)
raw_df_filtered['aqi'] = raw_df_filtered['aqi'].fillna(method='ffill')

import numpy as np
raw_df_filtered['aqi'] = raw_df_filtered['aqi'].replace('Unknown', np.
    nan)

# Apply forward fill to fill NaN values with the previous valid value
raw_df_filtered['aqi'] = raw_df_filtered['aqi'].fillna(method='ffill')

#Converting daylight and sunshine duration to hours
raw_df_filtered['daylight_duration'] = raw_df_filtered['
    daylight_duration'] / 3600
raw_df_filtered['sunshine_duration'] = raw_df_filtered['
    sunshine_duration'] / 3600

import seaborn as sns
import matplotlib.pyplot as plt

group1 = ['daylight_duration', 'sunshine_duration']

plt.figure(figsize=(15,10))
sns.boxplot(data=raw_df_filtered[group1])
plt.xticks(rotation=90)
plt.title('Spread of data among sunshine and daylight durations')
plt.show()

temp = ['temperature_2m_max', 'temperature_2m_min']

plt.figure(figsize=(10,10))
sns.boxplot(data=raw_df_filtered[temp])
plt.xticks(rotation=90)
plt.title('Spread of data among CO')
plt.show()
```

```
group4 = group1+temp

plt.figure(figsize=(15,10))
sns.boxplot(data=raw_df_filtered.drop(columns=(group4+['weather_code',
    'latitude', 'longitude'])))
plt.xticks(rotation=90)
plt.title('Spread of data among remaining features')
plt.show()

# Rename columns for better readability
raw_df_filtered.rename(columns={
    'date': 'Date',
    'county': 'County',
    'latitude': 'Latitude',
    'longitude': 'Longitude',
    'temperature_2m_max': 'Max Temperature',
    'temperature_2m_min': 'Min Temperature',
    'uv_index_max': 'Max UV Index',
    'wind_speed_10m_max': 'Max Wind Speed',
    'wind_gusts_10m_max': 'Max Wind Gusts',
    'aqi': 'Air Quality Index',
    'daylight_duration': 'Daylight Duration',
    'sunshine_duration': 'Sunshine Duration',
    'weather_code': 'Weather Condition'
}, inplace=True)

raw_df_filtered.head()

raw_df_filtered.to_csv('2022_2024_combined_weather_data.csv')
```

Listing 1: Data Cleaning Code

Below is the code used for the Flask web application:

```
import os
from flask import Flask, jsonify, request, send_file,
    render_template_string
from flask_cors import CORS
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
import openmeteo_requests
import requests_cache
import pandas as pd
from retry_requests import retry
from shapely import wkt
import geopandas as gpd
import random
import folium
import os
import warnings

app = Flask(__name__)
CORS(app)
warnings.filterwarnings("ignore")
data_frame = pd.read_csv('../dataset/cleaned_data/2022
    _2024_combined_weather_data.csv')
data_frame['Date'] = pd.to_datetime(data_frame['Date'])

```

```
cache_session = requests_cache.CachedSession('.cache', expire_after=3600)
retry_session = retry(cache_session, retries=5, backoff_factor=0.2)
openmeteo = openmeteo_requests.Client(session=retry_session)

# List of coordinates for each MA county
ma_counties_coordinates = [
    {"county_name": "Barnstable", "latitude": 41.7003, "longitude": -70.3002},
    {"county_name": "Berkshire", "latitude": 42.3118, "longitude": -73.1822},
    {"county_name": "Bristol", "latitude": 41.7938, "longitude": -71.1350},
    {"county_name": "Dukes", "latitude": 41.4033, "longitude": -70.6693},
    {"county_name": "Essex", "latitude": 42.6334, "longitude": -70.7829},
    {"county_name": "Franklin", "latitude": 42.5795, "longitude": -72.6151},
    {"county_name": "Hampden", "latitude": 42.1175, "longitude": -72.6009},
    {"county_name": "Hampshire", "latitude": 42.3389, "longitude": -72.6417},
    {"county_name": "Middlesex", "latitude": 42.4672, "longitude": -71.2874},
    {"county_name": "Nantucket", "latitude": 41.2835, "longitude": -70.0995},
    {"county_name": "Norfolk", "latitude": 42.1621, "longitude": -71.1912},
    {"county_name": "Plymouth", "latitude": 41.9880, "longitude": -70.7528},
    {"county_name": "Suffolk", "latitude": 42.3601, "longitude": -71.0589},
    {"county_name": "Worcester", "latitude": 42.4002, "longitude": -71.9065}
]

county_weather_data = {}

# Function to fetch weather data for a given county
def fetch_weather_data(county_name, latitude, longitude):
    url = "https://api.open-meteo.com/v1/forecast"
    params = {
        "latitude": latitude,
        "longitude": longitude,
      "current": ["temperature_2m", "relative_humidity_2m", "apparent_temperature", "precipitation", "weather_code", "wind_speed_10m", "wind_direction_10m", "wind_gusts_10m"],
      "daily": ["weather_code", "temperature_2m_max", "temperature_2m_min", "sunrise", "sunset", "uv_index_max", "precipitation_probability_max", "wind_speed_10m_max", "wind_gusts_10m_max"],
        "timezone": "America/New_York",
        "temperature_unit": "fahrenheit",
        "wind_speed_unit": "mph",
        "precipitation_unit": "inch",
        "forecast_days": 7
    }
```

```
    responses = openmeteo.weather_api(url, params=params)

    response = responses[0]

    current = response.Current()
    current_temperature_2m = current.Variables(0).Value()
    current_relative_humidity_2m = current.Variables(1).Value()
    current_apparent_temperature = current.Variables(2).Value()
    current_precipitation = current.Variables(3).Value()
    current_weather_code = current.Variables(4).Value()
    current_wind_speed_10m = current.Variables(5).Value()
    current_wind_direction_10m = current.Variables(6).Value()
    current_wind_gusts_10m = current.Variables(7).Value()

    current_data = {
        "time": [pd.to_datetime(current.Time(), unit="s", utc=True)],
        "temperature_2m": [current_temperature_2m],
        "relative_humidity_2m": [current_relative_humidity_2m],
        "apparent_temperature": [current_apparent_temperature],
        "precipitation": [current_precipitation],
        "weather_code": [current_weather_code],
        "wind_speed_10m": [current_wind_speed_10m],
        "wind_direction_10m": [current_wind_direction_10m],
        "wind_gusts_10m": [current_wind_gusts_10m],
    }

    current_df = pd.DataFrame(data=current_data)

    daily = response.Daily()
    daily_data = {
        "date": pd.date_range(
            start=pd.to_datetime(daily.Time(), unit="s", utc=True),
            end=pd.to_datetime(daily.TimeEnd(), unit="s", utc=True),
            freq=pd.Timedelta(seconds=daily.Interval()),
            inclusive="left"
        ),
        "county": county_name,
        "latitude": latitude,
        "longitude": longitude,
        "weather_code": daily.Variables(0).ValuesAsNumpy(),
        "temperature_2m_max": daily.Variables(1).ValuesAsNumpy(),
        "temperature_2m_min": daily.Variables(2).ValuesAsNumpy(),
        "sunrise": daily.Variables(3).ValuesAsNumpy(),
        "sunset": daily.Variables(4).ValuesAsNumpy(),
        "uv_index_max": daily.Variables(5).ValuesAsNumpy(),
        "precipitation_probability_max": daily.Variables(6).
  ValuesAsNumpy(),
        "wind_speed_10m_max": daily.Variables(7).ValuesAsNumpy(),
        "wind_gusts_10m_max": daily.Variables(8).ValuesAsNumpy()
    }
    daily_df = pd.DataFrame(daily_data)

    county_weather_data[county_name] = {
        "current": current_df,
        "daily": daily_df
    }
    # print(f"Processed weather data for {county_name}")

```

```python
for county in ma_counties_coordinates:
    fetch_weather_data(county["county_name"], county["latitude"], county["longitude"])

ma_counties_boundaries = pd.read_csv('../dataset/cleaned_data/ma_counties_boundaries.csv')

ma_counties_boundaries['geometry'] = ma_counties_boundaries['geometry'].apply(wkt.loads)

ma_counties_gdf = gpd.GeoDataFrame(ma_counties_boundaries, geometry='geometry')

def plot_heatmap(feature):
    feature_data = pd.DataFrame()
    for county, weather_data in county_weather_data.items():
        daily_data = weather_data["daily"]
        feature_data[county] = daily_data[feature]

    feature_data.index = daily_data["date"].dt.date

    if feature in ["temperature_2m_max", "temperature_2m_min"]:
        cmap = "coolwarm"
    else:
        cmap = "Purples"

    plt.figure(figsize=(12, 8))
    heatmap = sns.heatmap(feature_data.transpose(), cmap=cmap, annot=True, fmt=".1f", cbar=True)

    color_bar = heatmap.collections[0].colorbar
    if feature == "temperature_2m_max" or feature == "temperature_2m_min":
        color_bar.set_label('Temperature ( F )', rotation=270, labelpad=20)
    elif feature == "precipitation_probability_max":
        color_bar.set_label('Precipitation Probability (%)', rotation=270, labelpad=20)
    elif feature == "wind_speed_10m_max" or feature == "wind_gusts_10m_max":
        color_bar.set_label('Wind Speed (mph)', rotation=270, labelpad=20)
    else:
        color_bar.set_label(feature, rotation=270, labelpad=20)

    if feature == "uv_index_max":
        plt.title(f"7-Day UV Index Forecast by County", fontsize=16)
    else:
        plt.title(f"7-Day {feature.replace('_', ' ').title()} Forecast by County", fontsize=16)

    plt.xlabel("Date", fontsize=12)
    plt.ylabel("County", fontsize=12)
    plt.xticks(rotation=45)

    images_folder_path = os.path.join(os.path.dirname(__file__), '..', 'Frontend', 'static', 'images')

```

```
    os.makedirs(images_folder_path, exist_ok=True)
    plt.savefig(os.path.join(images_folder_path, f'{feature}_heatmap.
png'), bbox_inches='tight')

plot_heatmap("temperature_2m_max")
plot_heatmap("temperature_2m_min")
plot_heatmap("precipitation_probability_max")
plot_heatmap("wind_speed_10m_max")
plot_heatmap("wind_gusts_10m_max")
plot_heatmap("uv_index_max")

def plot_boxplot(feature):
    valid_features = [
        "temperature_2m_max",
        "temperature_2m_min",
        "sunrise",
        "sunset",
        "uv_index_max",
        "precipitation_probability_max",
        "wind_speed_10m_max",
        "wind_gusts_10m_max"
    ]

    if feature not in valid_features:
        raise ValueError(f"Invalid feature: {feature}. Please choose
from {', '.join(valid_features)}.")

    feature_data = []
    county_names = []

    for county, weather_data in county_weather_data.items():
        daily_data = weather_data["daily"]

        if feature in daily_data:
            feature_data.append(daily_data[feature])
            county_names.append(county)

    df = pd.DataFrame(feature_data).transpose()
    df.columns = county_names

    plt.figure(figsize=(12, 8))
    boxplot = sns.boxplot(data=df, palette="Set2")

    boxplot.set_title(f"Distribution of {feature.replace('_', ' ').
title()} by County", fontsize=16)
    boxplot.set_xlabel("County", fontsize=12)
    boxplot.set_xticklabels(county_names, rotation=45, ha="right")
    if feature == "temperature_2m_max" or feature == "
temperature_2m_min":
        plt.ylabel("Temperature ( F )", fontsize=12)
    elif feature == "wind_speed_10m_max" or feature == "
wind_gusts_10m_max":
        plt.ylabel("Speed (mph)", fontsize=12)
    else:
        plt.ylabel(feature.replace("_", " ").capitalize(), fontsize=12)

    plt.xticks(rotation=45)
    images_folder_path = os.path.join(os.path.dirname(__file__), '..',
```

```python
        'Frontend', 'static', 'images')
    os.makedirs(images_folder_path, exist_ok=True)
    plt.savefig(os.path.join(images_folder_path, f'{feature}_boxplot.png'), bbox_inches='tight')

plot_boxplot("temperature_2m_max")
plot_boxplot("temperature_2m_min")
plot_boxplot("precipitation_probability_max")
plot_boxplot("wind_speed_10m_max")
plot_boxplot("wind_gusts_10m_max")
plot_boxplot("uv_index_max")

@app.route('/')
def index():
    # Create a base Folium map centered around Massachusetts
    m = folium.Map(location=[42.4072, -71.3824], zoom_start=7, tiles="cartodbpositron")

    def random_color():
        return f'#{random.randint(0, 255):02x}{random.randint(0, 255):02x}{random.randint(0, 255):02x}'

    for _, row in ma_counties_gdf.iterrows():
        county_name = row['NAME']
        weather_info = county_weather_data[county_name]["current"]

        popup_text = (
            f"Temperature: {round(weather_info['temperature_2m'].values[0])} F <br>"
            f"Humidity: {weather_info['relative_humidity_2m'].values[0]}%<br>"
            f"Precipitation: {round(weather_info['precipitation'].values[0], 2)} in<br>"
            f"Wind Speed: {round(weather_info['wind_speed_10m'].values[0], 2)} mph<br>"
            f"Wind Direction: {round(weather_info['wind_direction_10m'].values[0], 2)}  <br>"
            f"Wind Gusts: {round(weather_info['wind_gusts_10m'].values[0], 2)} mph"
        )

        folium.GeoJson(
            row['geometry'],
            style_function=lambda feature, color=random_color(): {
                'fillColor': color,
                'color': 'black',
                'weight': 0.5,
                'fillOpacity': 0.6,
            },
            tooltip=folium.Tooltip(county_name),
            popup=folium.Popup(popup_text, max_width=300)
        ).add_to(m)

    map_html = m.get_root().render()

    return render_template_string('''{{ map_html|safe }}''', map_html=map_html)
```

```
@app.route('/weather', methods=['GET'])
def weather():
    county_name = request.args.get('countyName')
    info_types = request.args.get('typeOfInformation').split(',')
    from_date = request.args.get('fromDate')
    to_date = request.args.get('toDate')

    if not county_name or not info_types or not from_date or not to_date:
        return jsonify({"error": "Missing required parameters"}), 400

    filtered_data = data_frame[(data_frame['County'] == county_name) &
                               (data_frame['Date'] >= from_date) &
                               (data_frame['Date'] <= to_date)]

    if filtered_data.empty:
        return jsonify({"error": "No data found for the given criteria"}), 404

    result_data = []
    for info_type in info_types:
        if info_type not in data_frame.columns:
            continue
        series_data = filtered_data[['Date', info_type]].rename(columns={info_type: 'value'}).dropna()
        result_data.append({
            "info_type": info_type,
            "values": series_data.to_dict(orient='records')
        })

    return jsonify({
        "county_name": county_name,
        "data": result_data
    })

if __name__ == '__main__':
    app.run(debug=True)
```

Listing 2: Web Server code

# B Appendix B: Additional Figures

Include any additional figures or tables that support the analysis.

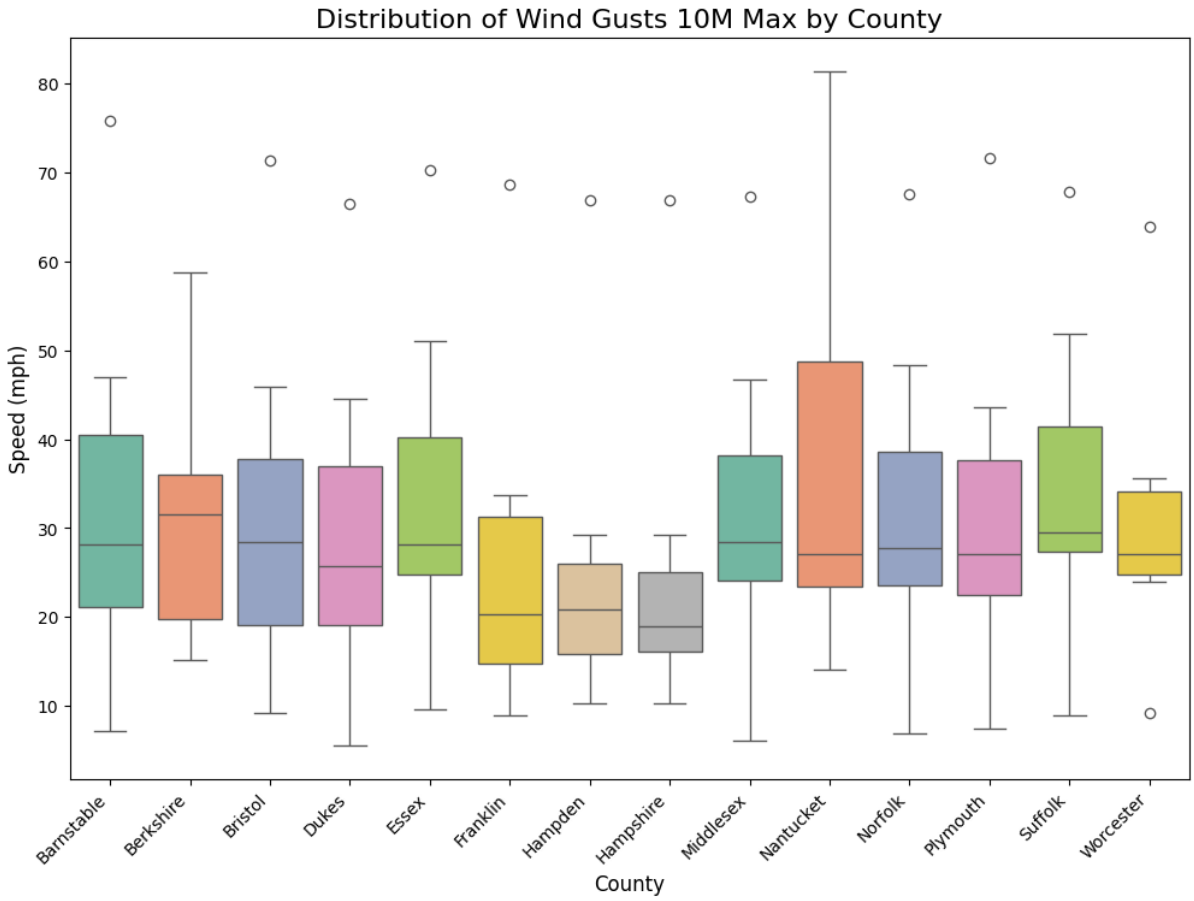


Figure 11: Wind Gust Box Plot