**GitHub: https://github.com/shailja-somani-0/ADS-507-Team-4**

**System Architecture: San Diego, California Weather Data and Alerts Pipeline**

Shailja Somani, Tara Dehdari, and John Vincent Deniega

Shiley-Marcos School of Engineering, University of San Diego

ADS 507: Practical Data Engineering

Professors Jonathan Sixt and Dillon Orr

February 24, 2024

# **Abstract**

Table of Contents

[Abstract 2](#_Toc159783371)

[Introduction 4](#_Toc159783372)

[Methods 4](#_Toc159783373)

[Initial SQL Database Configuration 4](#_Toc159783374)

[Data Extraction Methodology 7](#_Toc159783375)

[Daily API Calls 9](#_Toc159783376)

[Data Visualization Configuration 9](#_Toc159783377)

[Results 10](#_Toc159783378)

[Discussion 11](#_Toc159783379)

[References 12](#_Toc159783380)

# **Introduction**

Meteorology and weather – more broadly – can be observed in real-time or over specific periods of interest. In the case of significant weather events, these observations may prove vital to understanding the breadth of data applications ranging from understanding climate change to public safety (“Climate Models,” 2023). For example, in February 2024, California experienced powerful storms that caused major flooding, mudslides, and catastrophic damage (Bravo, 2024). As such, it is incumbent on domains including data science to utilize multiple sources of data, whether from data streaming sources such as thermometers and barometers or point observations recorded at hourly or daily intervals and to engineer data pipelines that provide useable data for end-users to analyze and make critical decisions.

Well-established sources such the National Weather Service [NWS], the University of California San Diego [UCSD], and OpenWeather together can inform the casual user and the data scientist with historical data analysis and weather and climate projections. So long as the data is thoughtfully treated and transformed to be compatible with each other and so long as they are standardized into a common schema, proper data engineering becomes the necessary conduit between these various weather data sources and the climate scientists and public safety officials seeking to find more effective solutions. Thus, we built out a data pipeline that combines data from the three aforementioned well-established sources into one SQL database that efficiently stores and summarizes (in a view) daily weather conditions and alerts, alongside historical weather data.

# **Methods**

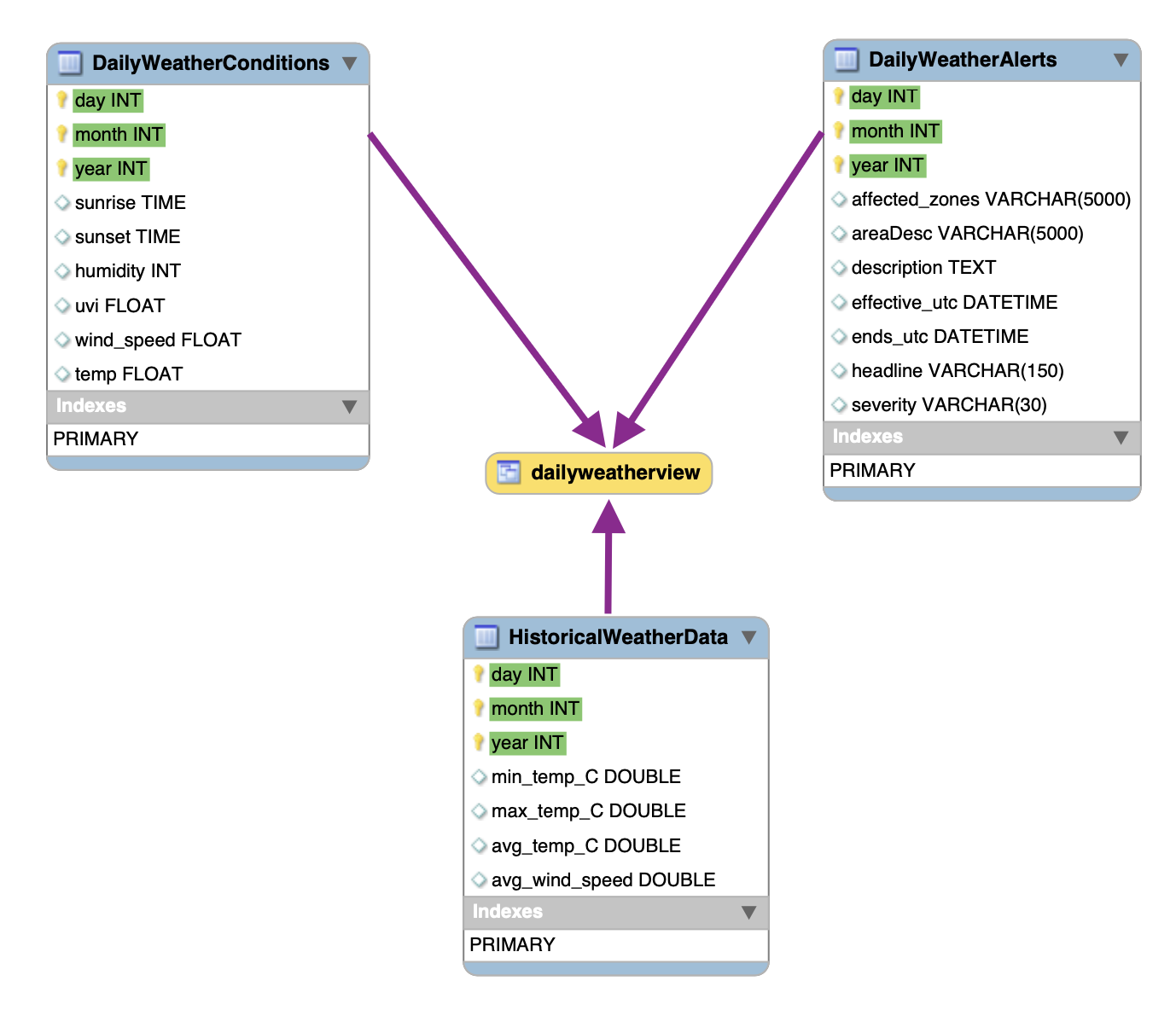
## **Initial SQL Database Configuration**

First, the ‘pymysql’ Python package allowed for an object to be created to take in network socket information as arguments to be passed to a MySQL server. This object acts as the interface in which embedded MySQL commands may be passed over Python such as selecting a database, passing MySQL syntax operations, or triggering execution of specific commands to the server. For this project, we connected to the local MySQL server on the machine of whomever executes the Python code. The server is directed to create a database called “WeatherDatabase” if it does not already exist, then create three tables if they do not exist: “DailyWeatherConditions,” “DailyWeather Alerts,” and “HistoricalWeatherData.” The day, month, and year columns for each table are integer values that cannot contain null values as the combination of these three fields will serve as the primary key for each table. However, we elected to keep the three columns as individual integers rather than only having a date primary key for easier joining across the two current daily tables and the historical table. By being able to join on just day and month across multiple years, that facilitates comparison across historical data. Throughout the tables, generally, any lists of geographic areas are stored as strings of VARCHAR of length 5000 as these lengths may vary depending on the quantity of alerts upon API call. Headline information is stored as a string of VARCHAR of length 150 as a reasonable length to capture the topic of the descriptive information. Description of the weather alert is stored as strings of TEXT as this narrative information contains verbose and extended data pertaining to the weather alert. Values such as humidity are stored as integer values as decimal values are not observed.

To the end of creating a singular view that contains daily (current day) data, as well as historical comparisons, the server is then directed to create a view called “DailyWeatherView.” This view combines the daily information from the “DailyWeatherConditions” and “DailyWeather Alerts” tables by joining on the entirety of the date (day, month, and year). Then, the daily data is joined to metrics from five and ten years ago, by joining to the “HistoricalWeatherData” table on day and month, as well as year minus five and year minus 10 (two joins are done in succession). The resulting view contains one row per day of data in the current year with columns for that day’s weather conditions and alerts, as well as historical weather conditions for five and ten years prior to that date. This view is very helpful for a daily snapshot tying together all three data sources, as well as to feed into visualizations, which we discuss further in the Results section. A schema diagram of our database is included below for clarity, as well as detailed information of all columns included in the DailyWeatherView.

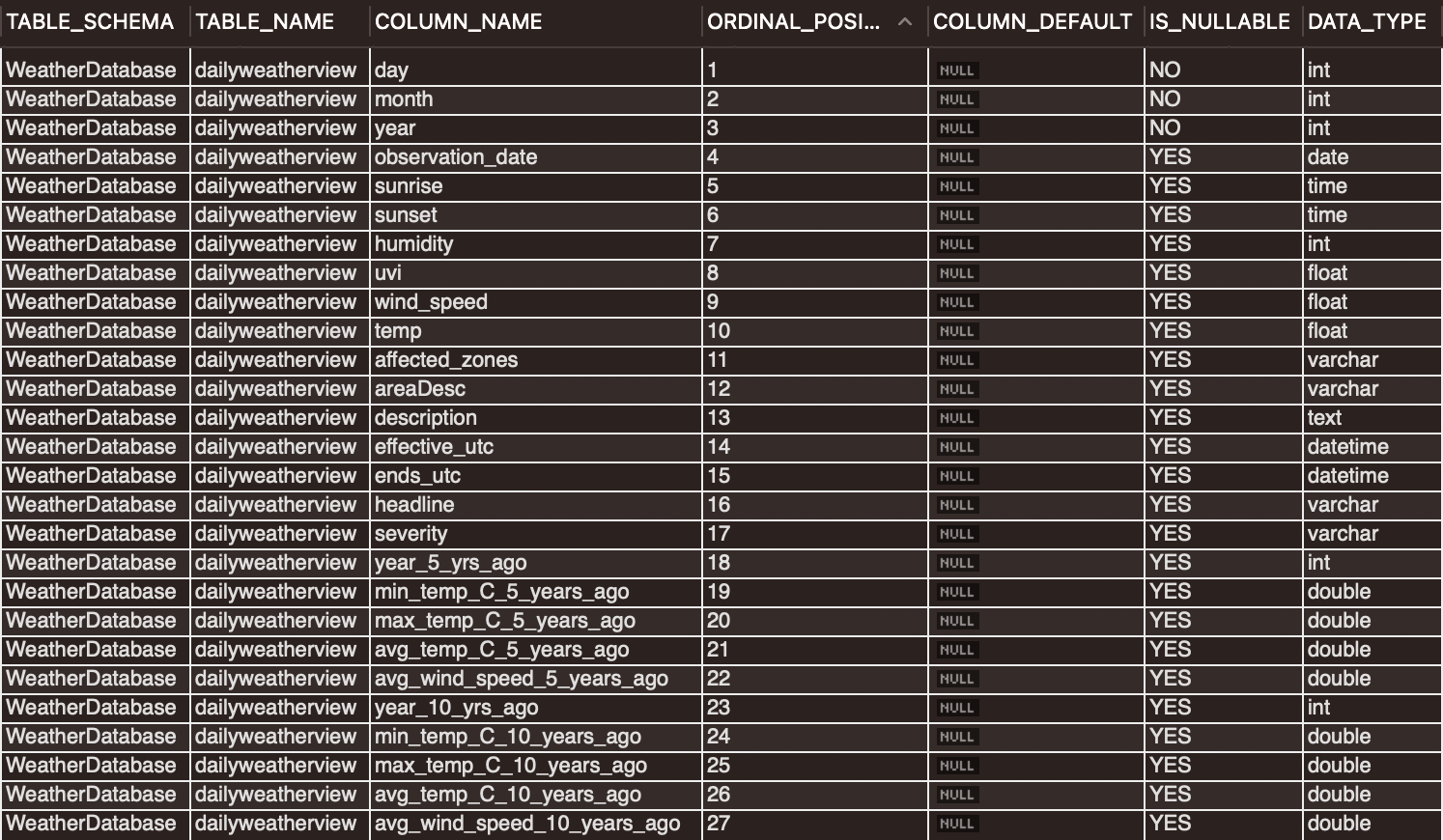
***Figure 1.***

WeatherDatabase Schema Diagram



***Figure 2.***

Detailed schema information for DailyWeatherView



## **Data Extraction Methodology**

With the MySQL database configured, data was extracted from three different sources. The ‘nwsapy’ Python package allowed interfacing with the official NWS API via Python language and associated methods and attributes. A user agent method established a connection to the NWS server by passing API and identification information as arguments to the user agent object (“API Web Service,” n.d.). Once the connection is established, the API is called to extract weather alerts for a given location. The “get\_alert\_by\_zone” method was called on the user agent object and was passed a six-character argument. This argument consisted of three A through Z characters followed by three zero through nine characters as a string value corresponding with the NWS’s identified geographic zones. An “AlertByZone” object is returned, which may be printed to standard output as either a dictionary or a Pandas DataFrame using the returned object’s respective methods.

However, not all geographic areas may have an active alert at the time the API is called, so the object that is returned to the user may contain a null value. In order to avoid inserting malformed null data into the MySQL database, this object is checked if the value is null, which if it is will set values in accordance with the target schema containing the primary keys used across all three sources: Date, Month, and Year. For this and the remaining values in accordance with the schema, null values and a headline of “No Alert Today” are staged in a string variable, which will be passed to the MySQL connection object for execution as an “INSERT INTO” statement.

The pipeline takes in a second data source, the UCSD historical weather dataset, in the form of a static comma-separated value [CSV] file. Data from this source contains daily observations of weather data from the period year 1939 to 2019, which was originally collected by the National Oceanic and Atmospheric Administration (“San Diego Weather,” n.d). Temperature data was transformed into the same units per the schema and dates were parsed into their respective day, month, and year values for primary key purposes. Columns that were valid, but did not have significant overlap or contextual importance with the other data sources were dropped, primarily consisting of temperature and wind speed flag information, of which the data was sparse in density. The transformed data was parsed into their respective schema variables and staged into a string variable containing the MySQL statement to be executed as an “INSERT INTO” statement.

The pipeline takes in the final data source, the OpenWeather API, which utilizes the ‘requests’ Python package to extract data via a uniform resource locator with an embedded latitude and longitude for San Diego, California as well as an assigned application identifier (“OpenWeather API,” n.d.). Similarly to the two sources above, data only from the current day is extracted, which is returned in a dictionary object. Data is identified in accordance with the target schema and staged into a string variable that is to be executed as an “INSERT INTO” statement, much like the previous sources.

## **Daily API Calls**

The initial configuration involved building the “WeatherDatabase” with three tables and one view, as well as dumping all the historical data into the “HistoricalWeatherData” table. It also involved two API calls to the NWS and OpenWeather APIs to load in data for the day the tables were all created. However, because that weather data is frequently updated and published, we elected to call both those APIs daily at 10am to insert the current day’s weather conditions and alerts into our database in real-time.

In order to automate the daily API calls, a separate Python file was created that first connects to the local MySQL server and selects the “WeatherDatabase.” Then, it calls both APIs and inserts the daily data isolated from them (as defined by what is in line with the schema) into the appropriate tables. This Python file is called by a daily CronJob (using Terminal on a Mac) that is set to run at 10am daily (Aleksic, 2023). As such, the pipeline extracts, by calling the respective API procedures, executes the transformation in the provided schema, and loads the data into the MySQL database using the embedded INSERT INTO statement, without any additional user effort.

## **Data Visualization Configuration**

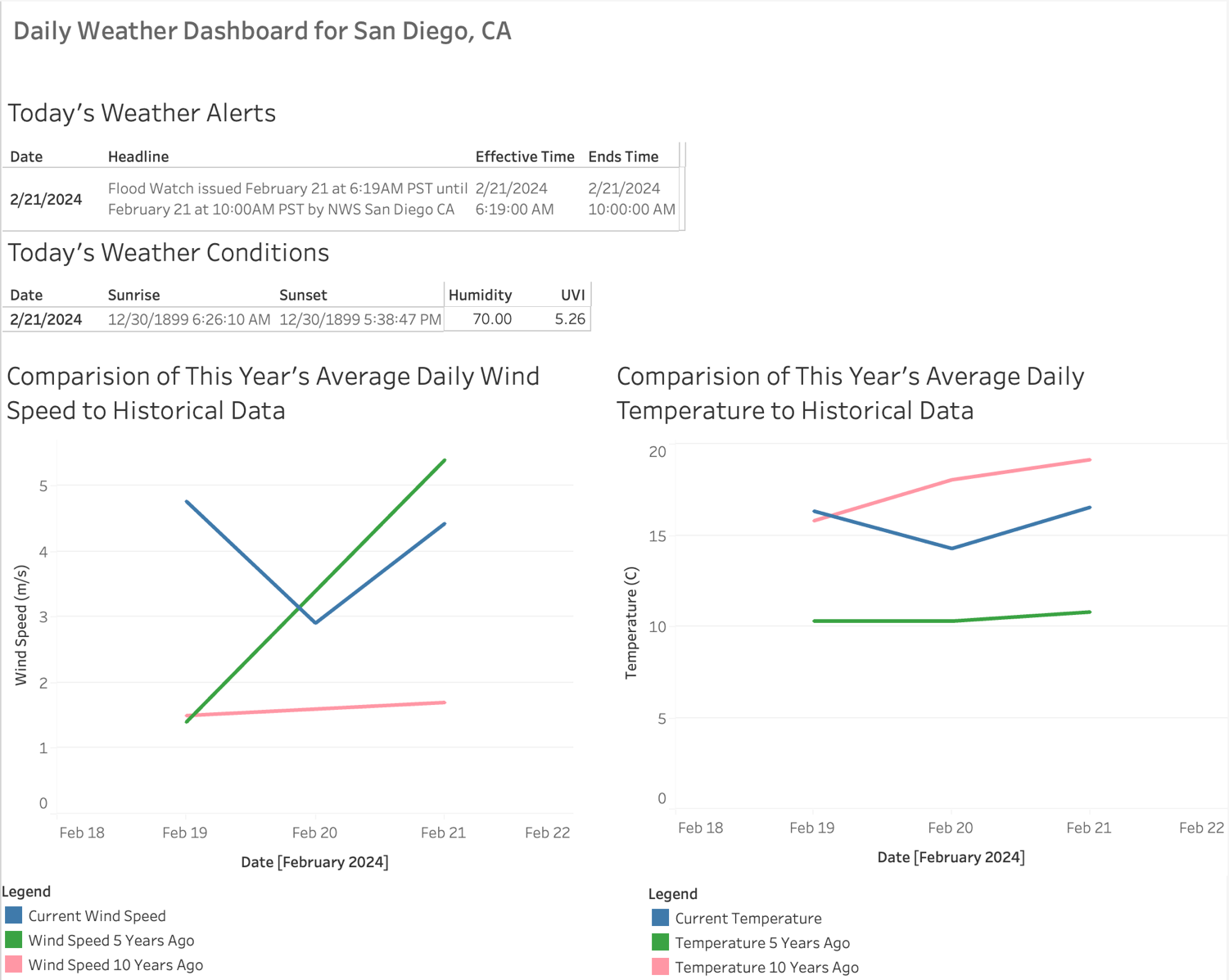
Given that the SQL database created (“WeatherDatabase”) will update daily with no foreseeable end point, we had to create visualizations in a way such that they would have a live connection to the data and be updated whenever the data is updated. To this end, we connected from Tableau Desktop to the local MySQL Server. To do so, we had to first install: (1) an ODBC driver and (2) the MySQL ODBC connector (“iODBC,” 2024 & “Connector/ODBC,” 2024). Following that, we were able to navigate to the “Connect to a Server” area within Tableau, select MySQL, enter our local server’s information and user credentials, then select “WeatherDatabase” to import. All three tables within the database and the view can be imported, but only the data in the view is required for our further work.

# **Results**

By creating a daily summary of data retrieved from all three data sources in the “DailyWeatherView,” then importing that into Tableau, we were able to create a brief, but informative, dashboard highlighting daily weather metrics, as well as a comparison to historical weather data. Readers should keep in mind that this dashboard should be seen as a prototype to demonstrate the functionality of our data pipeline rather than a full demonstration of visualization or analytics that can be supported with the pipeline. Our brief dashboard looks as follows, featuring key weather information for the current day at the top and graphs comparing this year’s weather trends to prior trends in the bottom two graphs. To show the dashboard at its full functionality, we elected to show a screenshot from February 21 because a weather alert was present that day. On days one is not present, the “Today’s Weather Alerts” table simply says “No Alert Today.”

***Figure 3.***

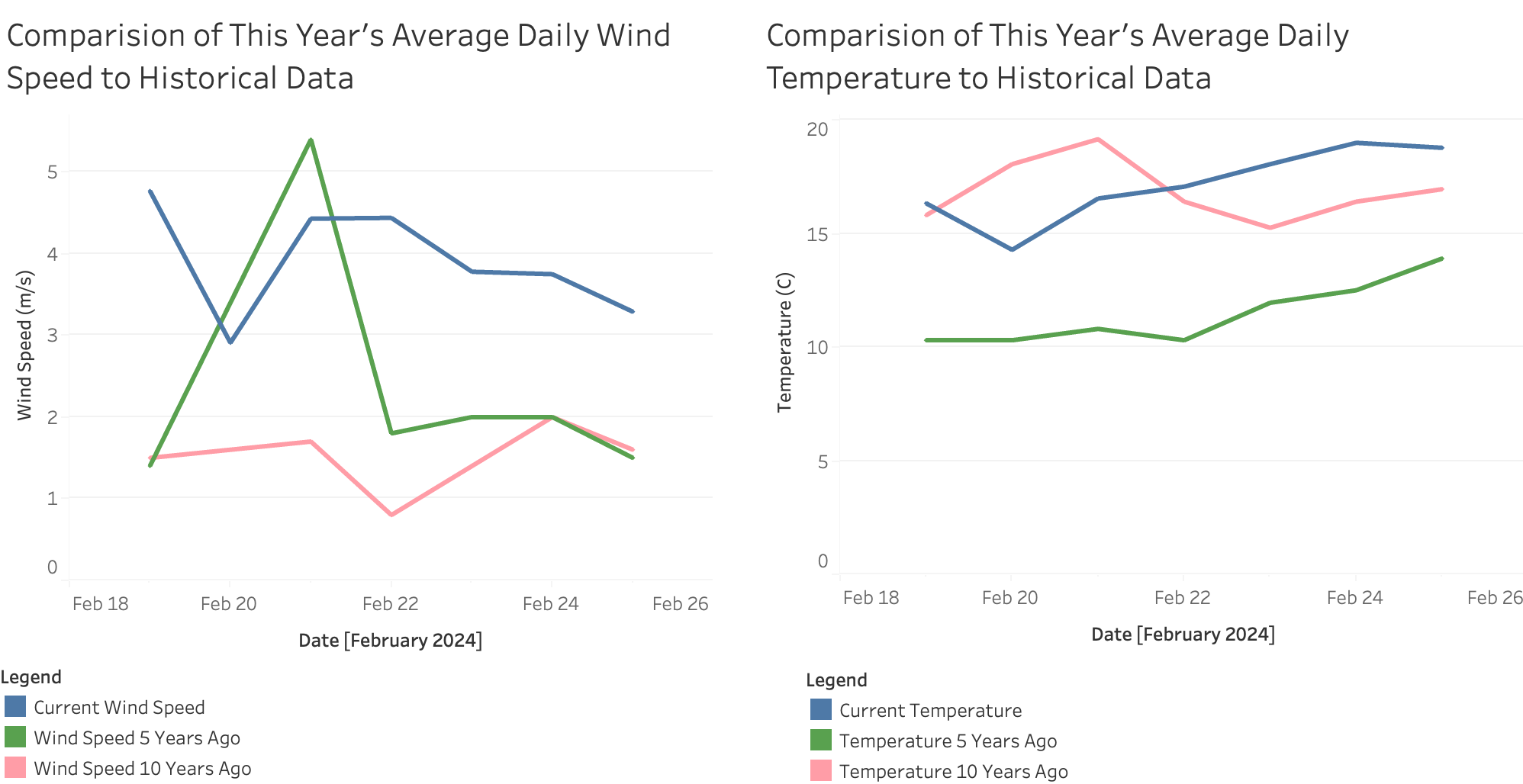
San Diego, CA Daily Weather Tableau Dashboard for February 21, 2024



The two tables at the top of the dashboard offer quick easily-digestible alerts and key data points for today’s weather to end users. The goal with those two tables is to provide a few key data points rather than overwhelm our users. If they wish to learn more about a specific alert, current weather conditions, etc, they may query the relevant table (“DailyWeatherConditions” or “DailyWeatherAlerts”). The graphs at the bottom provide users with quick, intuitive visualizations demonstrating weather trends this year and comparing those trends to five years ago and ten years ago. The color schemes are intentionally kept consistent across the two graphs so as to not confuse users about which year they are viewing data for. While alerts are only available some days, wind speed and temperature are available every day so, as our pipeline has continued to collect daily data, we have seen those two graphs flesh out more. Below is how the graphs looked on February 25, 2024.

***Figure 4.***

San Diego, CA Daily Weather and Wind Speed Graphs (via Tableau Dashboard for February 25, 2024)



The Tableau dashboard, as of February 25, 2024, is available on the Tableau public server at this link:<https://public.tableau.com/app/profile/shailja.somani/viz/Team4FinalProjectWeatherDashboard/Dashboard1?publish=yes>. However, because Tableau Public inherently does not allow us to give it permission to access our individual local MySQL server, it is simply displaying visualizations from a data extract created on February 25 rather than being updated live with the data source. However, the dashboard on our local machine does continue to update every day at 10am when our Cron job adds data into the SQL database. This Tableau dashboard allows non-technical users to quickly get a brief summary of daily weather conditions and alerts, along with comparison to historical trends, without having to individually access all three data sources feeding into the dashboard.

# **Discussion**

The data pipeline we built combines three independent data sources into a SQL database with one view summarizing daily data and including historical comparisons, as we as a Tableau dashboard featuring key information about the weather and any alerts available for today, as well as visual comparisons to historical weather trends. This allows both technical and non-technical users to quickly access vital weather information that impacts a variety of personal and business planning tasks every day. Furthermore, the automated daily Cron job allows for the data in the SQL database and visualizations to be easily updated without any technical resources needing to be allocated to it daily.

While we accomplished many objectives, due to time and resource constraints, there are some limitations to our pipeline as it exists. The largest limitation is that it is all on a local MySQL server rather than a cloud-based one due to difficulties reliably accessing a cloud-based server across multiple users. Because we do not have a persistent server running all of this, we must ensure that the laptop where the pipeline is built and maintained is turned on and awake at the time the Cron job is triggered daily. Furthermore, while our local Tableau dashboard updates every time the underlying data updates (given it is a live connection), we are unable to publish the connection to the data source to Tableau’s public server. Thus, anything published uses a static data extract and does not refresh every time our APIs our called via the Cron job. Finally, there is simply decreased collaboration and increased version control requirements when working with multiple people on individual local servers. Thus, if given more time to productionalize this data pipeline, moving it to a cloud-based SQL server would be our top priority.

In addition to limitations due to the SQL server being local, we also faced cost and storage limitations. In an ideal world, we would be calling the APIs much more frequently than daily to check for any changes in the daily weather alerts and metrics. In addition, we may have expanded it to all cities in California or perhaps even the entire United States, if given enough resources. However, we were limited due to: (a) storage constraints – the more data we retrieved, the more we would have to store locally and (b) cost constraints – the OpenWeather API that contains daily temperature, wind speed, humidity, etc information does have a cost based on the number of API calls made. Thus, there were cost limitations to how often and for how many cities we could make API calls, especially given how many we had to make while testing our pipeline. In a real-world setting, we would work with all our end users to identify the optimal refresh cadence, as well as the geographic areas we needed data for, and then request additional resources from both our team’s budget, as well as all end users’ budgets, to meet those requirements.

Alongside moving our database to the cloud and optimizing the refresh cadence and areas, next steps will include creating a comprehensive user guide for non-technical users (who may only ever interact with the dashboard) and technical users (who may choose to use the SQL database as well). Additionally, a quarterly survey will be sent out to all users of the SQL database and Tableau dashboard to determine if there are any areas for improvement. By implementing our existing data pipeline, alongside these next steps, we plan to create a robust, one-stop shop for daily weather conditions and alerts in San Diego, alongside valuable comparisons to historical weather data. By minimizing the time required for our end users to source all this data individually, we hope to facilitate more advanced analytics and key use cases for a variety of industries and public benefit agencies in San Diego.

# **References**

Aleksic, M. (2023, December 21) *How to Set Up a Cron Job on Mac.* PhoenixNap. Retrieved February 13, 2024, from <https://phoenixnap.com/kb/cron-job-mac>.

*API Web Service.* (n.d.) National Weather Service (NWS) National Oceanic and Atmospheric Administration (NOAA) Retrieved February 1, 2024, from <https://www.weather.gov/documentation/services-web-api>.

Bravo, Christina. (2024, February 21) *San Diego weather: Pacific storm brings heavy rain overnight, clear skies ahead*. MSN. Retrieved February 23, 2024 from <https://www.msn.com/en-us/weather/topstories/san-diego-weather-pacific-storm-brings-heavy-rain-overnight-clear-skies-ahead/ar-BB1iuha2>.

*Climate Models*. (2023, August 25) NOAA. Retrieved February 23, 2024 from <https://www.climate.gov/maps-data/climate-data-primer/predicting-climate/climate-models>.

*Connector/ODBC*. (2024) MySQL Community Downloads. Retrieved February 13, 2024, from <https://dev.mysql.com/downloads/connector/odbc/>.

*Independent Open DataBase Connectivity (iODBC) for Linux, MacOS X and Unix systems*. (2024) iODBC.org. Retrieved February 13, 2024, from <https://www.iodbc.org/dataspace/doc/iodbc/wiki/iodbcWiki/Downloads#Mac%20OS%20X>.

*San Diego Weather - GHCN Daily.* (n.d.) The Scripps Institution of Oceanography at University of California, San Diego (UCSD). Retrieved February 1, 2024, from <https://oceaninformatics.ucsd.edu/datazoo/catalogs/ccelter/datasets/13>.

*Weather API.* (n.d.) OpenWeather. Retrieved February 1, 2024, from <https://openweathermap.org/api>.