# SkySynth: Interactive Weather Data Platform

Brendan Kelley
CPSC 445-WI – Capstone Project

**Abstract**

*SkySynth is an interactive and secure web platform designed to provide real-time and customizable weather data. By leveraging various data sources, including historical, current, and forecasted values, the platform will allow users to obtain detailed weather information based on specified latitude and longitude coordinates. The project aims to integrate advanced technologies such as machine learning, software development, and modern web tools to create a comprehensive and user-friendly service.*

## Introduction

The goal of this project is to create an interactive and secure website that provides real-time values for weather variables that are customizable. Given a latitude and longitude, the website will output a range of current, historical, and forecasted data values.

## Tools and Technologies

### Website Development

- **AWS EC2:** Cloud hosting service for deployment.
- **PHP:** Server-side scripting for dynamic content.
- **MySQL:** Database management for user data and weather data storage.
- **Python 3:** Data processing and backend calculations.
- **LATEX:** Used for formatting calculations, proposals, and other ideas
- **Apache2:** Web server software.
- **ChatGPT4/3:** AI for debugging, formatting, (it formatted this proposal...)

### Calculations

- **Python 3:** Used for complex data analysis and integration.
- **C (GCC-8):** Utilized for performance-intensive calculations.
- **1962/1966 Standard Atmosphere:** Model standards for atmospheric calculations.
- **Open-Meteo API:** API for fetching weather data.

### Hardware

- **ML Tool (Phelps):** Machine learning tool for custom model development.
- **RTL-SDR (GQRX):** Software-defined radio for receiving weather satellite data.
- **Raspberry Pi 4B+:** Low-power computing device for remote data processing.
- **PC (5700XT / 6-Core AM4 Processor):** Main computing setup for development and data processing.

# Methodology

Using PHP and SQL, an end-user can register and sign in. After signing in, it creates an active user session. Once a user is signed in, they can input multiple latitude and longitude coordinates or click a "Find My Location" button to start retrieving data. Using the Open-Meteo API, the request will input the latitude, longitude, and other selected weather variables to retrieve the corresponding data from the NOAA GFS model. Associated weather graphs will be generated either from the PC, Raspberry Pi, or the ML tool and sent to the user's homepage.

# Additional Ideas

1. **Advanced Forecasting Model:** Using the ML tool and defined standards to create a model that more accurately forecasts data for a given latitude and longitude on an unknown time basis. Various genetic ML algorithms will be tested, using Voronoi polygons for interpolation, and training with realistic synthetic data ( 95% match of historical/forecast data).
2. **Satellite Data Integration:** Utilizing an RTL-SDR (1.766 GHz max) and GQRX on a PC to receive NOAA weather updates and satellite images from NOAA satellites (NOAA 15, 18, 19). If the ML tool is unsuccessful, this will serve as an alternative method for data collection.

# Collection of Formulas and Notes for Calculating Weather Values

This collection assumes the use of the 1962 STATMOS and 1966 SUPL. STATMOS models.

## Constants (Valid up to 90 km)

- $p_0$: Standard atmosphere at sea level – $1.2250\,\mathrm{kg/m^3}$
- $R^*$: Universal gas constant – $8.314\,32\,\mathrm{J\,K^{-1}\,mol^{-1}}$
- $G_0$: Gravitational constant at sea level – $9.806\,55\,\mathrm{m/s^2}$
- $M_0$: Mean molecular weight of air – 28.9644 (dimensionless)
- $T_i$: Absolute temperature of ice under one standard atmosphere ($p_0$) – $273.15\,\mathrm{K}$
- $a$: Equatorial radius of Earth – $6\,378\,178\,\mathrm{m}$
- $b$: Polar radius of Earth – $6\,356\,798\,\mathrm{m}$
- $GM$: Mass of Earth multiplied by gravitational constant – $3.986\,221\,6 \times 10^{14}\,\mathrm{m^3/s^2}$
- $J$: Second harmonic coefficient – 1.623495E-3 (dimensionless)
- $\gamma$: Ratio of specific heat of air at constant pressure – 1.40 (dimensionless)

## Names of Other Variables

- $\psi$: Geocentric latitude (usually 45.5425°)
- $p$: Density
- $P$: Pressure
- $r$: Radial distance from Earth (altitude)
- $\theta$: Longitude
- $\phi(g)$: Geopotential (acceleration due to gravity at $r, \psi$ (assumed $\psi$ for 1966))
- $T_m$: Molecular scale temperature in absolute thermodynamic scales
- Geopotential Height: $H$
- Geometric Height: $Z$

## Formulas

**1962 STATMOS Formula for $\phi(g)$**

$$\phi(g) = -\left(\frac{GM}{r}\right)\left[1 - \frac{J}{3}\left(\frac{a}{r}\right)^2\left(3\sin^2(\psi) - 1\right)\right] + \left(\frac{GM}{a}\right)\left(1 + \frac{J}{3}\right)$$

**1962 STATMOS Formula for Density**

$$p = \left(\frac{M_0}{R^*}\right)\left(\frac{P}{T_m}\right)$$

**1962 STATMOS Formula for Speed of Sound**

$$C_s = \sqrt{\gamma\left(\frac{R^*}{M_0}\right)T_m}$$

**1966 SUPL. STATMOS Formula for $\phi(g)$**

$$\phi(g) = 9.780356\left(1 + 0.0052885\sin^2(\phi) - 0.0000059\sin^2(2\phi)\right)\ \mathrm{m/s^2}$$

where $\phi$ is altitude, assumes $\psi$ for lat.

**1966 SUPL. STATMOS Formulas for $H$ and $Z$**

$$H = \left(\frac{G_0}{\phi(g)}\right)\left(\frac{r\,Z}{r + Z}\right)$$

$$Z = \frac{r\,H}{\left(\frac{G_0\,r}{\phi(g)}\right) - H}$$

# System Architecture Diagram

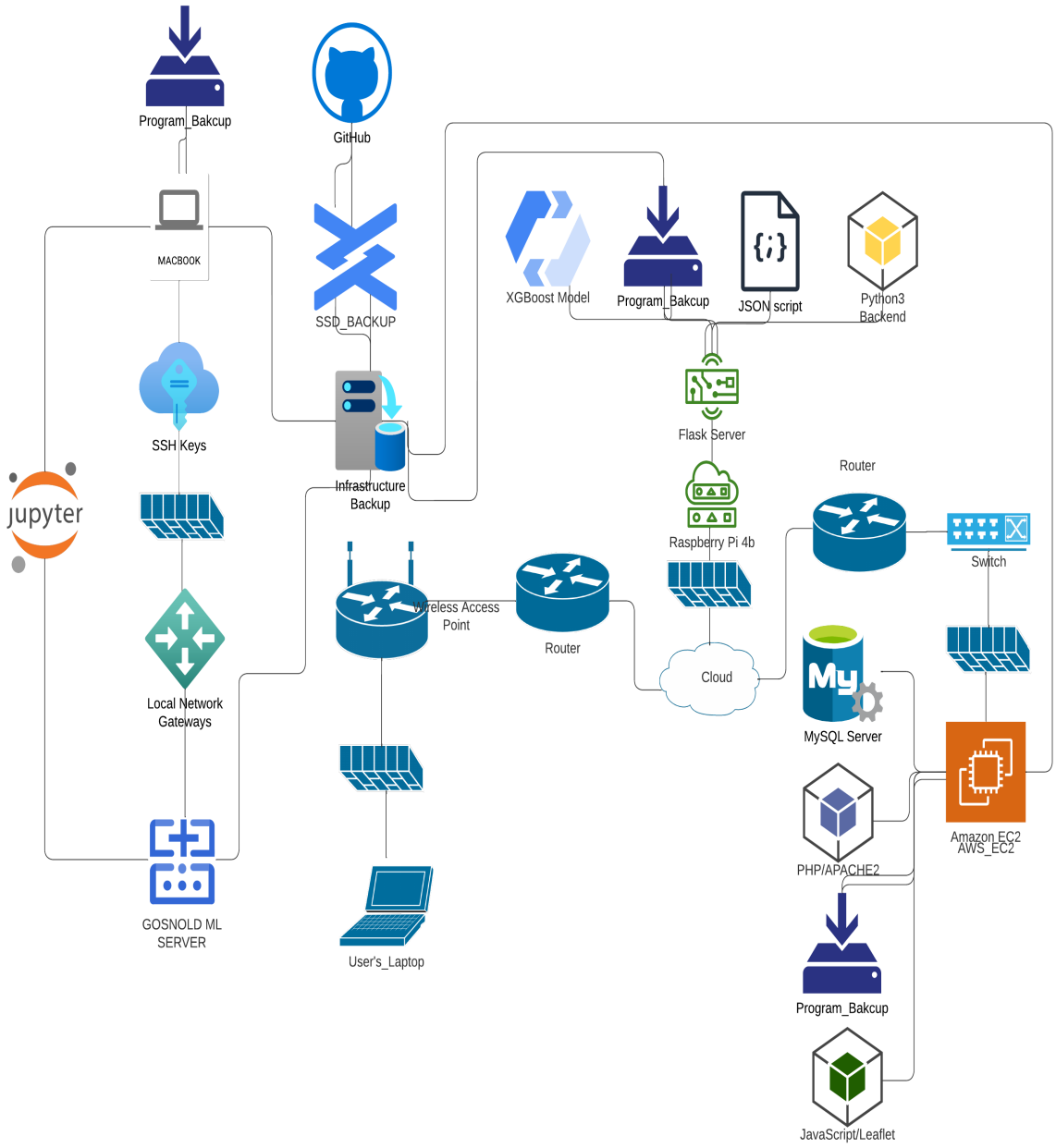


Figure 1: System Architecture Diagram

# Weather Forecasting Techniques

To predict the weather given multiple values over a time period, you can apply several forecasting methods depending on the type of data you have and the desired accuracy of the prediction. Here are some of the common methods and techniques used for weather forecasting:

# 1. Time Series Analysis Methods

Time series methods are statistical techniques that model the patterns and trends in historical data to forecast future values.

## ARIMA (Auto-Regressive Integrated Moving Average)

ARIMA is a widely used statistical method for time series forecasting. It captures patterns such as trends, seasonality, and cycles in the historical data. ARIMA is suitable for univariate time series data where only past observations are used to predict future values.

## Seasonal Decomposition of Time Series (STL)

This method decomposes the time series data into seasonal, trend, and residual components. It's useful for understanding the underlying patterns in the data and for making short-term predictions when seasonality is strong.

## Exponential Smoothing (ETS)

ETS models apply exponential weights to past observations, where more recent observations have a greater impact on the forecast. Variants of this method, such as Holt-Winters Seasonal Smoothing, account for trends and seasonality.

# 2. Machine Learning and AI Techniques

Machine learning methods are useful when dealing with large datasets with multiple variables (features) and complex patterns.

## Linear Regression

This method predicts the future value of a variable by modeling the relationship between the dependent variable (e.g., temperature) and one or more independent variables (e.g., humidity, pressure).

## Random Forests and Gradient Boosting

These ensemble learning techniques build multiple decision trees to make more accurate predictions. They are effective in capturing non-linear relationships between variables and handling high-dimensional data.

## Neural Networks

Neural networks, particularly **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM) networks**, are powerful for time series forecasting. They are designed to handle sequential data and can learn complex temporal dependencies, making them well-suited for weather prediction.

## Support Vector Machines (SVM)

SVMs can be used for regression (SVR) in weather prediction, especially when the data shows non-linear relationships. They can be trained to find the best hyperplane that represents the relationship between input variables and the output.

## 3. Numerical Weather Prediction (NWP) Models

Numerical weather prediction uses mathematical models based on the laws of physics, such as the Navier-Stokes equations, to simulate the atmosphere's behavior.

### Weather Research and Forecasting Model (WRF)

WRF is a widely used NWP model that relies on the initial conditions of the atmosphere and computes the evolution of weather elements over time. It requires significant computational power and access to real-time atmospheric data.

### Global Forecast System (GFS)

GFS is a global-scale NWP model run by the National Weather Service in the U.S. It predicts weather conditions up to several days in advance and is used by meteorologists worldwide.

## 4. Hybrid Approaches

Combining different methods can often yield better predictions. For example, combining a physical NWP model with machine learning techniques can correct biases in the NWP model's outputs, providing more accurate forecasts.

## Steps to Apply Forecasting Techniques

1. **Collect and Prepare Data:** Gather historical weather data, such as temperature, pressure, humidity, and wind speed, for your area of interest. Ensure the data is clean, formatted correctly, and free of missing values.
2. **Analyze the Data:** Visualize the data to identify trends, patterns, and seasonality. Use statistical analysis to understand the correlations between different variables.
3. **Choose a Forecasting Method:** Select a suitable method based on the data type, the length of the forecasting horizon, and the computational resources available.
4. **Build and Train the Model:** Split your data into training and testing sets. Use the training data to build and train the model, adjusting parameters to optimize performance.
5. **Validate the Model:** Evaluate the model's accuracy using the testing data and metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or R-squared.
6. **Deploy the Model for Forecasting:** Once validated, deploy the model to make real-time predictions.

## Tools and Libraries for Weather Forecasting

- **Python Libraries:** `statsmodels` for ARIMA, `scikit-learn` for machine learning models, `Keras` or `TensorFlow` for neural networks, and `pandas` for data manipulation.
- **Specialized Software: WRF** or **GFS** for NWP modeling.
- **Data Sources:** Access weather data from APIs like the OpenWeatherMap API, NOAA, or meteorological services in your country.

# Current Website Abilities and Ongoing Work – 9/10/24

## Current Website Abilities

- Ability to register
- Ability to sign in
- Main home page and other pages created
- Logout functionality

## Currently Working On

- SMS text message API for password recovery using VONAGE API
  - If SMS doesn't work, will use lockout questions
- Finishing settings, help, and data requests front end
- Finalized standards for analysis

## What I need Done

- Ability to pull in data through API
- Create Python3 analysis tools
- Send and display graphs / value from analysis
- Add other basic website functionality

# 1 Temporal / Seasonal / Climatological Conditions)

Understanding the temporal, seasonal, and climatological patterns is essential for atmospheric studies and weather forecasting. The following conditions were observed for September 26:

- **El Niño / La Niña:** The El Niño Southern Oscillation (ENSO) phenomenon significantly impacts global weather patterns. During an El Niño event, the central and eastern Pacific experience warmer-than-average sea surface temperatures, leading to altered precipitation and temperature patterns across the globe. Conversely, La Niña brings cooler-than-average sea surface temperatures, affecting trade winds and jet stream positioning.
- **Seasonal Conditions:** As we approach the autumnal equinox, the Northern Hemisphere begins to cool, marking a transition from summer to fall. Seasonal shifts impact weather systems, and the reduction in solar radiation causes a weakening of summer's intense heat.
- **Atlantic Isotherm at 35° Lat, 75° Lon (September):** Around this latitude and longitude, particularly during late September, the Atlantic Ocean surface temperature begins to decline, reducing the strength of tropical systems in the region. This area typically marks a boundary between warmer tropical waters and cooler temperate regions.
- **Northeast (NE) Trade Winds:** These winds are a key feature of the tropical atmosphere, blowing from the northeast toward the equator. They play a crucial role in steering tropical storms and cyclones and influence the flow of moisture and weather systems from the Atlantic.
- **Northeast (NE) Jet Stream:** The NE jet stream, located in the upper atmosphere, becomes more active during the transition from summer to fall. It can influence the path of mid-latitude cyclones and storms, bringing cooler air from the polar regions down into the United States and modifying the temperature patterns across the East Coast.

Climatological Summary

The current conditions are heavily influenced by the ENSO phase, with the possibility of a developing El Niño. The NE trade winds and jet stream exhibit seasonal variability, shaping weather outcomes in the Atlantic region. The gradual decline in sea surface temperatures near 35° Lat, 75° Lon marks the end of the peak hurricane season, while the Northeast jet stream becomes more active, introducing colder air masses from higher latitudes.

# Week 2 Log: 10/1/24

Research Activities

1. **Statistical Approaches:**
   - Investigating statistical methods for data analysis, with a specific focus on the KS test for synthetic data.
2. **Causality Exploration:**
   - Considering the development of a custom ML algorithm for assessing causality.
   - Initial checks will be conducted online for existing methodologies.
3. **Update on RR:**
   - After reviewing RR, it seems different methods will need to be applied to various datasets.
   - Notably, it performs decently at predicting temperature over 24 hours but less effectively for precipitation.
   - This aligns with the understanding that temperature doesn't fluctuate as significantly as rain on a daily basis.
   - It's essential to consider time periods, particularly using hourly data for both temperature and precipitation forecasting.
4. **Data Collection Needs:**
   - Gather hourly data from Open-Meteo, focusing on granularity from at least four different weather stations around the target latitude and longitude.
   - This approach will enable bilinear and bicubic interpolation for historical data specific to certain lat/lon coordinates.
   - Aiming to collect historical weather data across various locations in Virginia, enhancing the ML tool's predictive capabilities.
5. **Visualization Tools:**
   - Explore `Folium` and `Geopy` for displaying collected data visually.
6. **Improvements for `rr.py` (Ridge Regression):**
   - Enhance the model by adding more predictor columns, highlighting the need for additional data.
   - Experiment with different rolling horizons for better accuracy.
   - Consider changing the model to use XGBoost or Random Forest for improved performance.
7. **NOAA Station Data Collection:**
   - Compile the locations of all NOAA stations in Virginia for comprehensive data collection.
   - Collect both daily and hourly records from Open-Meteo at Christopher Newport University's latitude and longitude.
   - Utilize WORLDMET (despite its R-based format) to gather NOAA ISD data, including potential historical ISD data.

# Week 3 Log: 10/8/24

Research Activities

1. **Research on Forecasting Models:**
   - Investigating Ridge Regression and ARIMA time series forecasting for weather data prediction.
   - Assessing the potential of XGBoost and Random Forest models to improve performance for predictive weather models.
2. **3D LiDAR Mapping:**
   - Spent 2.5 hours creating 3D LiDAR maps using USGS .laz files for the Hampton area.
   - Generated both heatmap and 3D scatter plot visualizations of the LiDAR data.
3. **Data Collection:**
   - Set up a cron job to run a shell script for data collection from Open-Meteo.
   - The script allows for 5% NaN values in the dataset and collects data every 24 hours.
   - Continuing to rely on Open-Meteo due to the NCEI (Asheville) outage caused by a hurricane.
4. **Statistical Testing:**
   - Began researching the KS test for comparing distributions and generating synthetic data for training models.
5. **Exploration of Interpolation Techniques:**
   - Reviewing bilinear and bicubic interpolation methods for weather data analysis to improve accuracy in historical data collection.
6. **Data Source Limitations:**
   - NOAA's NCEI API remains down, limiting access to historical data and forcing a temporary reliance on Open-Meteo.
   - Exploring alternative data sources for historical weather data.

# Week 3 Log: 10/8/24

Research Activities

1. **Research on Forecasting Models:**
   - Investigating Ridge Regression and ARIMA time series forecasting for weather data prediction.
   - Assessing the potential of XGBoost and Random Forest models to improve performance for predictive weather models.
2. **3D LiDAR Mapping:**
   - Spent 2.5 hours creating 3D LiDAR maps using USGS .laz files for the Hampton area.
   - Generated both heatmap and 3D scatter plot visualizations of the LiDAR data.
3. **Data Collection:**
   - Set up a cron job to run a shell script for data collection from Open-Meteo.
   - The script allows for 5% NaN values in the dataset and collects data every 24 hours.
   - Continuing to rely on Open-Meteo due to the NCEI (Asheville) outage caused by a hurricane.
4. **Statistical Testing:**
   - Began researching the KS test for comparing distributions and generating synthetic data for training models.
5. **Exploration of Interpolation Techniques:**
   - Reviewing bilinear and bicubic interpolation methods for weather data analysis to improve accuracy in historical data collection.
6. **Data Source Limitations:**
   - NOAA's NCEI API remains down, limiting access to historical data and forcing a temporary reliance on Open-Meteo.
   - Exploring alternative data sources for historical weather data.

# Python Ridge Regression Backtesting and XGBoost Training

**Overview**

This section describes two processes:

1. **Ridge Regression Backtesting:** A Python-based function that tests a Ridge regression model across different time periods using weather data.
2. **XGBoost Training and Evaluation:** A machine learning pipeline for training an XGBoost model to predict surface pressure based on weather features, including early stopping, cross-validation, and model saving.

## 1. Ridge Regression Backtesting

**Objective:** To evaluate a Ridge regression model by backtesting it on historical weather data, optimizing the alpha parameter, and tuning model performance based on mean absolute error (MAE).

**Key steps:**

1. Define new columns, target variable, and data file paths.
2. Implement a backtesting function that fits the model on training data and tests it on a rolling basis for a specified time period.
3. Optimize the model's performance by adjusting the Ridge regression's alpha hyperparameter.
4. Evaluate model predictions using MAE and stop optimization once the error difference falls below a predefined threshold (min_delta).
5. Use multiprocessing to process each weather data header in parallel, utilizing all available CPU cores.

```
def backtest(data, model, predictors, start=365*5, step=70):
    ...
    return pd.concat(all_pred)

def process_header(current_value):
    ...
    return f"Best MAE for {current_value}: {best_mae:.6f} with optimal alpha: {best_alpha}"
```

**Result:** The script prints the best alpha and MAE for each weather header processed.

## 2. XGBoost Model Training and Prediction

**Objective:** To build and evaluate a predictive model for surface pressure using the XGBoost library. The model uses historical weather data to learn relationships between input features and surface pressure.

**Key steps:**

1. Load the dataset and process feature and target arrays.
2. Perform a train-test split to evaluate the model's performance.
3. Train an XGBoost model using early stopping to avoid overfitting and use cross-validation to further assess model performance.
4. Save the trained model to a file for later use.
5. Load the saved model and predict the surface pressure for new weather data.

```
# Define hyperparameters
params = {"objective": "reg:squarederror", "tree_method": "hist"}

# Train the model with early stopping
```

```
model = xgb.train(
    params=params,
    dtrain=dtrain_reg,
    num_boost_round=200,
    evals=evals,
    verbose_eval=5,
    early_stopping_rounds=50
)
```

**Result:** After training, the script saves the model to a file and evaluates it on the test dataset, calculating the root mean squared error (RMSE) to quantify performance. The model can be loaded later for prediction on new data.

**5-fold Cross-Validation:** The model also undergoes cross-validation to ensure its robustness. The best RMSE obtained from cross-validation is recorded for further analysis.

## 3. Predictions on New Data

**Objective:** To demonstrate how the trained model can be applied to new weather data.

**Key steps:**

1. Prepare the new data in a format that matches the training data.
2. Use the saved model to make predictions for the new data points.
3. Print the predicted surface pressure for validation.

```
# Create DMatrix for the new data
new_dmatrix = xgb.DMatrix(new_data, enable_categorical=True)

# Make predictions with the loaded model
predictions = loaded_model.predict(new_dmatrix)
```

**Result:** The script prints the predicted surface pressure for the new weather data provided.

# Documentation for Generating user heat maps

**Date**: November 7, 2024

The code is focused on obtaining weather data for a user-specific geographic area, creating weather prediction models, and generating heatmaps from predictions.

## 1. Initial Imports

The code begins with importing several Python libraries:

- **matplotlib.pyplot** and **seaborn**: Used for plotting visualizations and creating statistical graphics.
- **numpy**: A core library for numerical computations, especially working with arrays.
- **openmeteo_requests**: For interacting with the Open-Meteo API to fetch weather forecasts.
- **requests_cache**: Caches HTTP requests to avoid repetitive API calls, speeding up data fetching.
- **pandas**: Essential for data manipulation and handling time series or tabular data.
- **retry_requests**: Implements retries for failed HTTP requests with exponential backoff to improve data fetching reliability.
- **os**: Handles operating system-level operations, such as reading files or managing directories.
- **PIL (Python Imaging Library)**: Used for image processing, notably for creating PNG files from heatmap data.
- **scipy.interpolate**: Provides tools for interpolation (e.g., filling in missing values or smoothing data).
- **geopandas**: Helps manage and analyze geographical data (though it's not directly used in this script).
- **argparse**: Facilitates command-line argument parsing to allow for dynamic input by the user.
- **xgboost**: Implements gradient boosting models for prediction tasks, particularly used in the script for weather prediction.

## 2. Caching and Retry Setup

The script sets up two key components for handling HTTP requests:

- **cache_session**: A cached session for the `requests` library that stores responses for one hour (3600 seconds) to speed up repeated requests.
- **retry_session**: Extends `cache_session` with retry logic, automatically retrying failed requests up to five times with a backoff factor of 0.2.

## 3. Function Definitions

### get_bound_area(lat, lon)

This function generates a square region around a given latitude and longitude. It calculates four corners (top-left, top-right, bottom-left, bottom-right) based on a central point, then visualizes this bounding area using a scatter plot.

### grab_4_points(lat, lon)

This function creates a list of four corners surrounding a given point. These corner points are essential for creating weather prediction models at different locations around a user's specified point.

### create_user_models(user_ID, lat, lon, filt)

This is the core function of the script. It creates weather prediction models based on historical weather data and the coordinates of five points: the center point and the four surrounding points. For each of these five points, the function:

- Loads a pre-trained XGBoost model.
- Retrieves historical weather data.
- Makes weather predictions for the next 15 days (hourly).
- Saves the results in CSV files.

This function returns a list of future prediction data for each point.

### create_heat_maps(user_ID, lat, lon)

Once the predictions are made for the five points, this function generates heatmaps from these predictions. The heatmaps represent the weather data in a visual format. The function then combines these individual heatmaps into a GIF, which can be used on the SkySynth website for user display. The heatmaps are generated hourly, and the old ones are deleted after the GIF is created to save storage.

### create_gif_from_pngs(source_dir, output_gif_path, duration=150)

This helper function takes PNG images from a given directory, sorts them, and converts them into a GIF. It's primarily used in the `create_heat_maps` function to generate a GIF from hourly weather prediction heatmaps.

## 4. Main Script Execution

The main execution block begins by setting up argument parsing using `argparse`. It takes four arguments:

- `user_ID`: A unique identifier for the user.
- `lat`: Latitude of the user's location.
- `lon`: Longitude of the user's location.
- `filt`: The specific weather filter for prediction (e.g., temperature, precipitation).

Once the arguments are parsed, the function `create_user_models` is called to generate the models and predictions. The execution time of the script is then printed.

## 5. Data Flow and Purpose

- **Location-based Data**: The functions primarily focus on gathering and predicting weather data for locations specified by the user. The user's latitude and longitude define the geographic area for weather predictions.
- **Model Predictions**: XGBoost models, trained on historical weather data, are used to predict the weather at each of the five points around the user's location.
- **Heatmap Visualization**: The predictions are visualized as heatmaps to make the weather data more interpretable. These heatmaps are then converted into a GIF, which can be used for user display on a website.

## 6. Conclusion

This script combines several key functionalities: weather prediction, data visualization (heatmaps), and automated GIF creation. It retrieves data from the Open-Meteo API, performs machine learning predictions using a pre-trained model, and generates visually appealing heatmaps. By combining these elements, the script provides a comprehensive tool for making detailed weather forecasts for specific geographic areas.

## Heat Map example

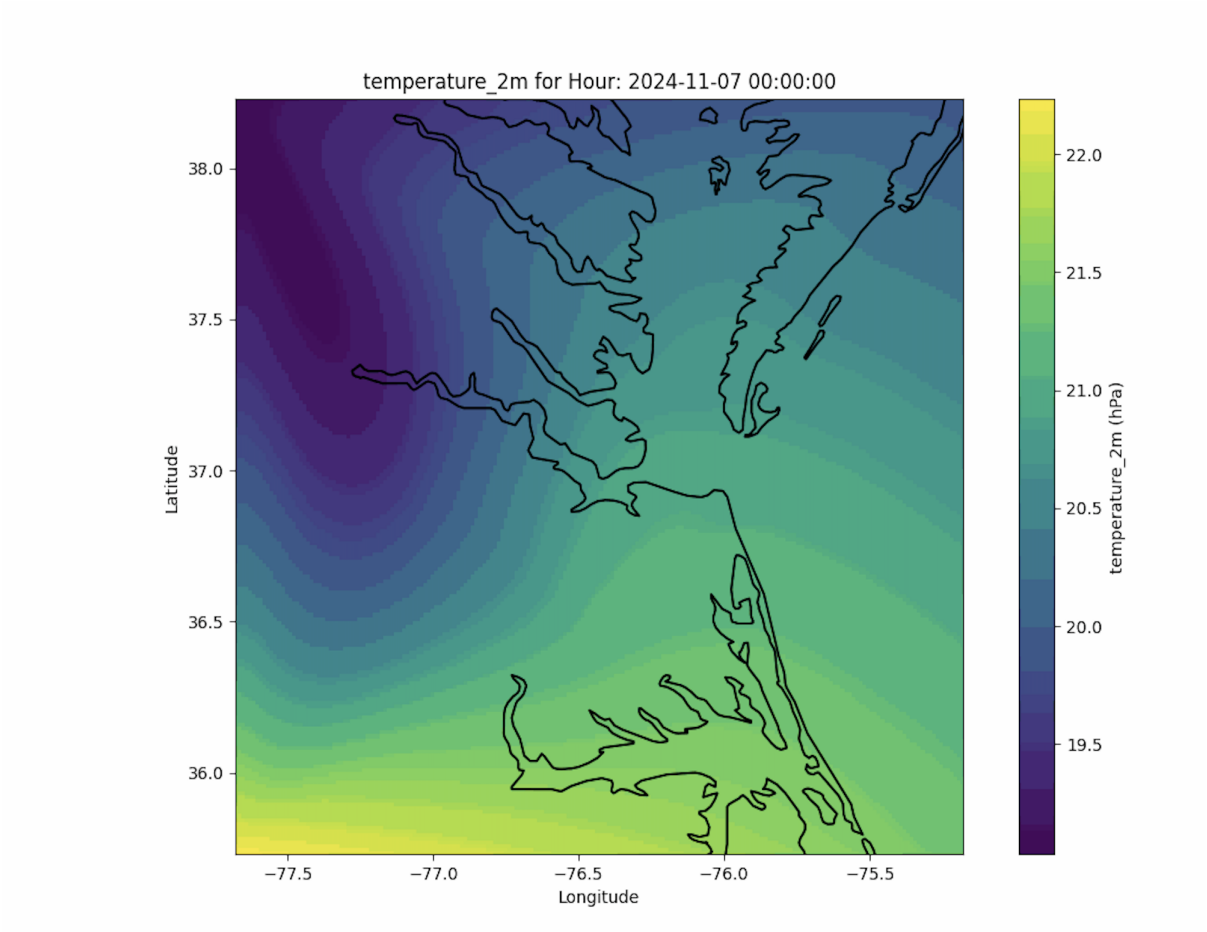


Figure 2: Heat Map Over CNU

# PHP Code for Home page

**Date**: November 7, 2024

## Overview

PHP code for a web-based application that allows users to interact with a map and save their selected location coordinates. The code includes features for user authentication, database interaction, and map integration with Leaflet.js.

## 1. Session and Database Connection

The script starts by initiating a session with `session_start()` and establishing a MySQL database connection using the `mysqli` library.

- The connection is checked, and if it fails, an error message is displayed.
- The user's email is retrieved from the session, and their corresponding `User_ID` is fetched from the database.
- Latitude and longitude values for the user are also fetched from the `User_Lat` and `User_Lon` tables based on the `User_ID`.

## 2. HTML and Map Integration

The HTML structure contains the following key elements:

- A `div` with the ID `map` that is used to display the map using Leaflet.js.
- A top navigation bar with links to other pages such as `Data Requests`, `Resources`, `Account Settings`, and `Logout`.
- A greeting message that displays the user's email if they are logged in, and an option to log in if not.

## 3. JavaScript for Map Interaction

- Leaflet.js is used to display the map, with a default location centered at coordinates $[37, -76.4]$.
- Users can click on the map to select a location, and the latitude and longitude are displayed on the page and sent to the server via an `XMLHttpRequest`.
- The `saveCoordinates` function sends the coordinates to the `save_location.php` script to save them in the database.

## 4. PHP Code for User Authentication

The PHP script checks if the user is signed in by verifying the session variable `User_ID`. If the user is signed in, their email is displayed along with a message inviting them to click on the map to select their location.

## 5. Conclusion

This script allows users to interact with a map, select a location, and save the coordinates. It integrates session handling for user authentication, database interaction for storing user locations, and JavaScript for map interaction.

# Leaflet Picture

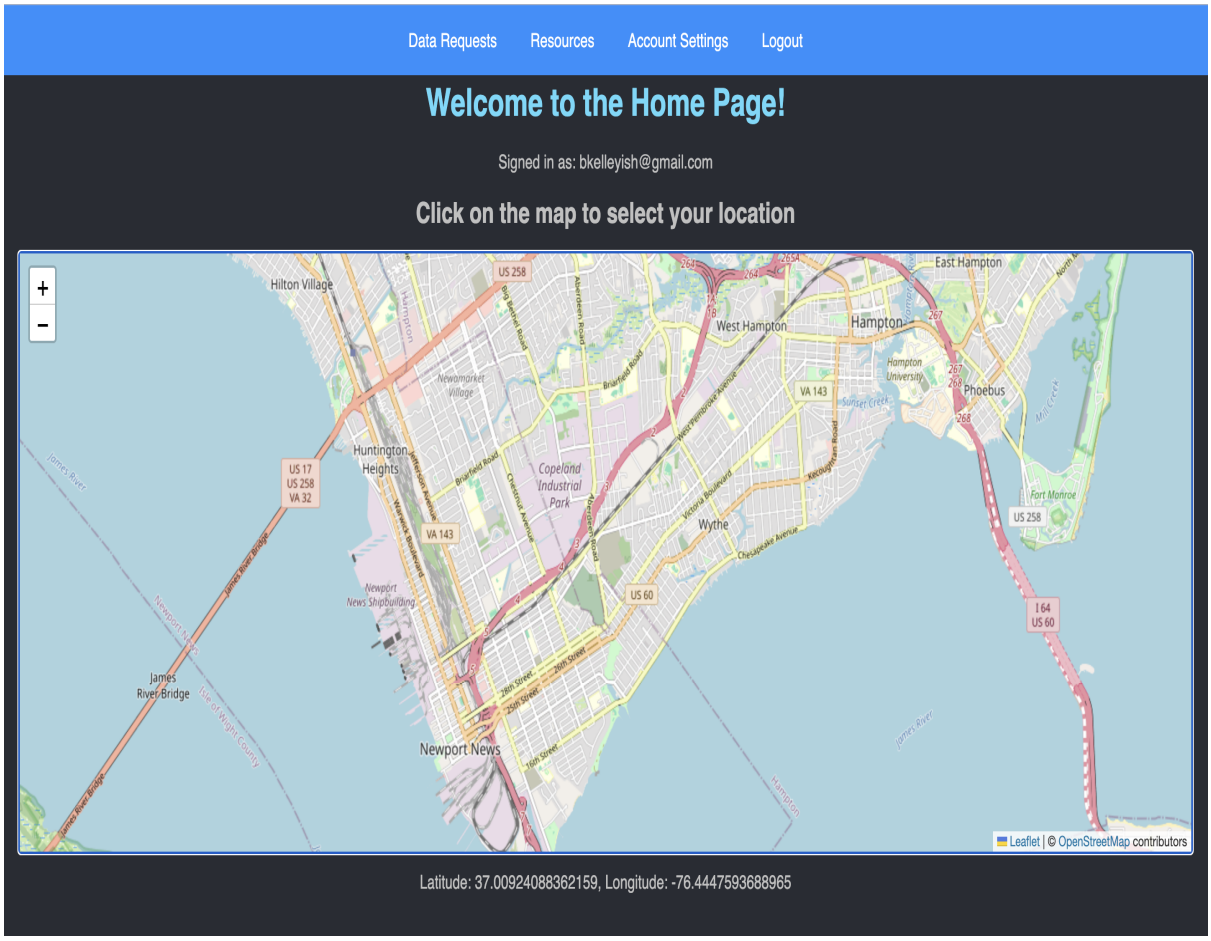


Figure 3: Leaflet.js in action