****

Software Engineering Department  
 ORT Braude College

Capstone Project Phase B – 61999

**WeatherGuide - An Application For Long-range Weather Forecasting Project**

**Authors:**

Foze Abo Zhaya – Foze.a.z177@gmail.com

Jasmen Mura – Jasmenmura@e.braude.ac.il

**Supervisor:**

Dr. Zakharia Frenkel

**24-2-D-32**

**Link to Github :https://github.com/Jasminmura/Capstone-Project.git**

**Table Of Contents**

[Abstract](#_fqwwe8rx2nxg)

[1. Introduction](#_kw3ti34n01e4)

[Problem Statement](#_vmb33ecnb5o4)

[Existing Solutions](#_d7kddgsz0egy)

[Proposed Solution](#_gidn7nkl875c)

[System Goals](#_yix1vikw3mya)

[Stakeholders and Benefits](#_ecd5pinmlyhe)

[2. Expected Achievements](#_89xqhmi70sce)

[2.1 Did We Meet the Project Benchmarks?](#_uezp42rqbl4m)

[3. Models Developed](#_du57427jacd3)

[3.1 Location Search Model](#_cmpjhuwh6rzv)

[3.2 Data Processing and Preprocessing Model](#_jeecoi471zzl)

[3.3 Clustering and Prediction Model](#_z0chsivrkj2b)

[3.4 Prediction Model](#_esepif1ekbqm)

[3.5 Backend and Frontend Models](#_g4qltkwkp4hh)

[3.6 Summary](#_s3h42tbqr60r)

[4. Engineering Process](#_le5gauroavwi)

[4.1 Process](#_61fbvuj6g2ch)

[4.1.1 Process Overview](#_nq2a1ygvg18)

[4.1.2 Detailed Steps of the Engineering Process](#_mbwlalkl4iph)

[4.1.3 Constraints and Challenges](#_che8ci6cbcjk)

[4.2 Product](#_wetvkvxd7xbj)

[4.2.1 Algorithms Used](#_f9wcn2xv5mh0)

[Nearest Neighbor Search:](#_5hifc0g1hqj1)

[K-Means Clustering:](#_qbejj1qr9m0a)

[Data Normalization:](#_zbf91anq8x71)

[Historical Pattern Matching:](#_dwm47694bjmw)

[4.2.2 Why These Algorithms Are Effective](#_c5xqhuslbrwu)

[4.2.4 Requirements](#_3pvik6bq6qye)

[4.2.5 GUI](#_xazgg2pooffo)

[4.2.5 Diagrams   
4.2.5.1 Use Case Diagram](#_80itiq1vjpgm)

[4.2.5.2 Activity Diagram](#_1q50krqmmsi2)

[5. Tests](#_vlic3hrojpcy)

[6. Challenges and Solutions](#_xcdc96586g5j)

[6.1. Handling large Datasets](#_5oa4jt6py9ns)

[6.2. Efficiently Finding the Nearest Location](#_r3bojetxup9y)

[7. Results and Conclusions](#_2xde60wxvf55)

[7.1 How We Addressed Challenges:](#_l6sjn7x30i4n)

[7.2 Decision-Making Considerations:](#_i4ufeljkuk8o)

[7.3 Lessons Learned](#_jy6ypcbxoiny)

[8. User Guide](#_8kh44q29ypxr)

[1.Home Page:](#_pmb4sh3rk1w6)

[1. Forecast Form Section](#_d8muiy1kcvwa)

[2. Pick a Location Section](#_molp62hxhur3)

[3. Forecast Overview Section](#_1hjdnhb4hmzo)

[2. About Page Explanation](#_og1lm0j9i9tn)

[9. Maintenance Guide](#_d6ae6r8mdfut)

[9.1. High-Level Overview](#_2vmb5i8df1g)

[9.2. Backend Components (Flask + Python)](#_90dtuyjorviv)

[9.2.1. Data\_loader.py:](#_cn1eifweyxwz)

[9.2.2 App.py:](#_w8s8rvz48wmt)

[9.2.3 Kmeans\_logic:](#_vh8qrvjzsf4u)

[9.3. Frontend Components (React)](#_25i8krl1xac)

[9.3.1 ForecastForm](#_qh4upz91ekv1)

[9.3.2 MapComponent](#_k2f3hzkszp9z)

[9.3.3 App.js](#_acww7x0vjuf)

[9.3.4 ForecastResult](#_88fb0dusuw9i)

[9.3.5 Navigation & About Page](#_7n995a8r7c0f)

[10. References](#_350w9pwwrsll)

# **Abstract**

Accurate long-range weather forecasting is critical for sectors like agriculture, event planning, and disaster preparedness, where strategic decision-making depends on reliable predictions. While many scientific groups develop advanced long-term forecasting models, a robust platform to deliver actionable results to users remains absent. Existing solutions often focus on short-term predictions, leaving industries without practical tools for long-term planning.

This project bridges that gap by developing a software application that sources prediction results from high-resolution ERA5 reanalysis data and advanced machine learning techniques. ERA5 provides a comprehensive global dataset of historical weather observations and model data, forming the foundation for reliable long-term forecasts. The application leverages clustering algorithms to analyze historical patterns and calculate the probabilities of multiple possible weather scenarios, offering forecasts that extend weeks or even months ahead.

The user-friendly interface will visually present key forecast data, including temperature, rainfall probabilities, and cloud coverage, tailored to user needs. Users will be able to explore how current conditions compare with historical patterns through intuitive visualizations, simplifying complex data interpretation.

In its initial phase, the project focuses on documenting the design, methodologies, and technologies to lay a solid foundation for full implementation. Once complete, the project will provide a much-needed tool to bridge the gap between scientific forecast models and practical user applications, enabling industries to plan and prepare more effectively.

# **Introduction**

Long-range weather forecasting involves predicting weather conditions over extended periods, typically several weeks to months. Accurate long-term forecasts are essential for sectors such as agriculture, travel, and event planning, where early preparation and informed decision-making are critical. However, most existing forecasting models focus on short-term predictions (up to 10 days), leaving a significant gap in practical, long-term weather insights for users.

Traditional forecasting relies on Numerical Weather Prediction (NWP) models that simulate atmospheric processes using mathematical equations and extensive current weather data. While these models have improved significantly, their accuracy diminishes over time due to the complexities of atmospheric dynamics and data limitations. As a result, reliable tools for delivering long-range weather forecasts remain scarce.

Recent advances in data science, particularly deep learning, have shown promise in addressing these challenges. Techniques like artificial neural networks can analyze large, complex datasets and identify hidden patterns, making them ideal for long-term weather prediction. By combining these advanced methods with comprehensive meteorological data, forecasting accuracy and reliability can be enhanced.

This project seeks to bridge the gap by developing a software application that not only generates long-range forecasts but also effectively sources and presents prediction results to users. The platform will utilize high-resolution ERA5 reanalysis data, offering hourly estimates of global climate and weather from 1940 to the present. With its advanced clustering algorithms and probabilistic forecasting methodologies, the application will predict multiple potential weather outcomes weeks or months ahead. Through its intuitive interface, users will gain access to valuable insights tailored to their needs, providing them with the reliable long-term forecasts necessary for strategic planning and decision-making.

## Problem Statement

Despite advances in weather prediction, the increasing unpredictability of weather patterns continues to create challenges for sectors such as agriculture, travel, and event planning. These industries rely on accurate long-range forecasts to optimize operations, mitigate risks, and seize opportunities. However, current forecasting services primarily focus on short-term predictions, typically up to 10 days, leaving a critical gap in long-term insights that are essential for strategic planning. Without reliable long-range forecasts, users face inefficiencies, heightened risks, and missed chances for effective preparation and resource management.

While many scientific groups develop advanced long-term predictive algorithms, a robust platform to deliver these results to users is missing, making long-range forecasting inaccessible and underutilized by those who need it most.

## Existing Solutions

Current weather forecasting solutions rely heavily on short-term models that deliver accurate predictions within a week or 10 days. Some research initiatives provide seasonal forecasts, but these are often highly technical, difficult to interpret, and designed primarily for scientific purposes rather than practical everyday use. These solutions fail to provide the user-friendly, accessible platforms that non-specialists in sectors like agriculture and travel require. Additionally, most of them present broad averages instead of actionable, probability-based insights.

The lack of a system to effectively source and present prediction results further limits the adoption of long-term forecasting, leaving users without practical, actionable weather insights.

## Proposed Solution

Our project addresses the gap between scientific long-term weather prediction models and practical user needs by developing a robust software platform that sources and presents long-range forecast results. By leveraging high-resolution ERA5 reanalysis data and advanced clustering algorithms, the application will provide forecasts of potential weather scenarios weeks or even months ahead. Rather than offering single-point or average predictions, the platform focuses on presenting probabilistic forecasts, particularly for rain probabilities, which are critical for sectors like agriculture, travel, and event planning.

The application will feature intuitive visualizations and interactive charts, making it easy for users to interpret complex data. Users can select their preferred locations through an interactive map or search bar and receive personalized forecasts tailored to their needs. The forecasts will help users plan activities, events, and operations while minimizing weather-related risks.

## System Goals

Our system aims to address the gap between long-range weather forecasts and the ability of users to effectively understand and apply this information in their daily or long-term planning. While many scientific institutions develop advanced long-term forecasting models, their output is often not accessible or actionable for non-experts.

The main goals of the system are:

* To deliver long-term weather forecasts in a clear and user-friendly manner by presenting probabilistic forecasts that show a range of possible weather scenarios, rather than a single, deterministic prediction.
* To bridge the gap between scientific prediction models and practical applications by providing forecasts that users can easily interpret and apply to make informed decisions, whether for agricultural planning, event scheduling, or trip preparation.

By achieving these goals, the platform will transform complex weather data into actionable insights, ensuring that users benefit from advanced forecasting methodologies without needing in-depth meteorological expertise.

## Stakeholders and Benefits

**General Public:** Our application helps individuals plan outdoor activities, trips, and vacations with greater confidence by offering long-range weather forecasts, reducing the likelihood of weather-related disruptions.

**Event Planners:** By providing probabilistic long-term weather forecasts, event planners can better schedule and organize outdoor events, minimizing risks and optimizing event planning to avoid adverse weather conditions.

**Tour Operators and Outdoor Enthusiasts:** Tour operators, hikers, campers, and outdoor sports enthusiasts benefit from accurate long-range forecasts by choosing the best times for outdoor activities and reducing risks associated with sudden weather changes.

**Small Businesses (Outdoor Services):** Businesses like landscapers and construction services can schedule outdoor tasks more efficiently, avoiding delays or downtime due to unpredictable weather.

**Benefits of the Solution for Stakeholders**

**1. General Public**

How the Solution Helps:

* Planning Activities and Travel:  
  Long-range weather forecasts allow individuals to plan outdoor activities, vacations, and trips more effectively, reducing the risk of cancellations or disruptions due to unexpected weather.
* Reducing Weather Disruptions:  
  Reliable forecasts help people prepare for and minimize disruptions caused by extreme weather events, improving safety and comfort.

**2. Event Planners**

How the Solution Helps:

* Scheduling Events:  
  Event planners can select optimal dates for outdoor events, reducing risks from adverse weather such as rain or wind.
* Optimizing Contingency Planning:  
  With access to forecast probabilities, planners can prepare backup plans, ensuring event success regardless of changing conditions.

**3. Tour Operators and Outdoor Enthusiasts**

How the Solution Helps:

* Choosing Optimal Travel Dates:  
  Long-term weather insights help tour operators plan excursions or activities during periods of favorable weather.
* Minimizing Outdoor Risks:  
  Hikers, campers, and outdoor sports enthusiasts can reduce risks and enhance their experiences by avoiding unfavorable weather conditions.

**4. Small Businesses (Outdoor Services)**

How the Solution Helps:

* Efficient Task Scheduling:  
  Businesses offering outdoor services can optimize schedules, reducing the risk of delays due to adverse weather.
* Reducing Resource Waste:  
  Proper weather planning helps avoid wasted time and resources, ensuring projects are completed on time and within budget.

This structured breakdown ensures stakeholders understand how WeatherGuide supports decision-making and minimizes weather-related risks.

# **2. Expected Achievements**

In developing the Long-range Weather Forecasting Project, we aim to achieve several key objectives that will significantly impact various stakeholders and improve forecasting capabilities. Below is a detailed outline of the expected achievements:

**2.1. Development of an Accurate Long-range Forecasting Model**

Objective: Develop and implement a reliable model for predicting weather conditions weeks or months in advance.

Expected Achievements:Enhanced Prediction Accuracy: Improvement in the accuracy of long-term weather predictions by integrating advanced data science techniques such as deep learning and high-resolution reanalysis data from ERA5.

Identification of Weather Patterns:Ability to detect and forecast significant weather patterns and trends over extended periods.

**2.2.Creation of a User-friendly Software Application**

Objective:Build an accessible software platform that provides long-range weather forecasts to various user groups.

Expected Achievements:Intuitive User Interface: Design a user-friendly interface that allows easy access and interpretation of weather forecasts by the general public, farmers, and planners.

Customization Features: Include features for customizing the forecast display based on user preferences, such as specific locations, time ranges, and types of weather information.

Multi-platform Accessibility: Ensure the application is accessible across multiple devices, including web, mobile, and desktop platforms.

**2.3.Integration of High-resolution ERA5 Data**

Objective: Utilize ERA5 data to provide detailed and comprehensive weather insights.

Expected Achievements: Comprehensive Data Utilization: Leverage ERA5's detailed historical and current data to enhance the accuracy and reliability of forecasts.

Advanced Analysis: Enable sophisticated analyses such as uncertainty measurements and retrospective climate assessments to support long-range forecasting.

**2.4.Provision of Valuable Insights for Decision-making**

Objective: Offer actionable weather insights that support better planning and decision-making for various sectors.

Expected Achievements: Improved Planning: Provide long-term weather forecasts that assist users in planning agricultural activities, events, and travel more effectively, reducing the impact of adverse weather conditions.

Risk Management: Enable better risk assessment and management by forecasting potential extreme weather events well in advance.

Operational Efficiency: Enhance operational efficiency in sectors like agriculture and event planning by allowing stakeholders to anticipate and adapt to weather changes.

### **2.1 Did We Meet the Project Benchmarks?**

Yes, we successfully met the key benchmarks set for the project. Our primary goal was to develop a tool to source and present long-range weather prediction results effectively, and we achieved this by implementing accurate probabilistic forecasting using ERA5 data and K-Means clustering. The system’s core functionality, including location-based input, clustering, and forecast presentation, was completed as planned.

We also met our benchmarks for creating an intuitive user interface, ensuring users could easily access and interpret prediction results. While certain advanced goals, like market adoption and multi-platform deployment, were deferred, the system was designed to be scalable and maintainable for future enhancements. Overall, the project fulfilled its main objectives of accuracy, usability, and scalability.

# **3. Models Developed**

The development of WeatherGuide involved multiple interconnected models that work together to process data, identify weather patterns, and provide accurate forecasts through an intuitive interface. Each model plays a critical role in ensuring that the system remains efficient, scalable, and practical for users.

## 3.1 Location Search Model

The **Location Search Model** ensures that user-provided latitude and longitude inputs are accurately mapped to relevant weather data, even if the user’s exact location is not present in the dataset. This model begins by attempting to find an exact match for the provided coordinates in the system’s database of geographic locations. If no match is found, it calculates the nearest location using the **Nearest Neighbor Search algorithm** based on Euclidean distance. This design guarantees that users receive forecasts tailored to their geographical proximity, preventing gaps in data coverage and ensuring that the forecasts remain relevant.

## 3.2 Data Processing and Preprocessing Model

Once the appropriate location is identified, the Data Processing and Preprocessing Model takes over. This model ingests large CSV files containing historical weather data for temperature, rainfall, wind, and moisture spanning several years. The data is first cleaned, with any missing or inconsistent values handled through interpolation or averaging nearby values. After cleaning, the system normalizes all weather variables to a common scale to prevent one parameter (e.g., wind speed) from dominating others during analysis. The model also aligns the data across different time periods, ensuring that weather patterns from different years can be accurately compared during clustering. This careful preprocessing prepares the data for efficient and accurate clustering, which is essential to producing reliable forecasts.

## 3.3 Clustering and Prediction Model

At the core of the system lies the Clustering and Prediction Model, which utilizes the K-Means clustering algorithm to group historical weather data into clusters of similar conditions. The clustering process segments the weather data into distinct patterns, such as hot and dry conditions, rainy periods, or windy and humid scenarios. Each cluster is associated with representative statistics, including the average and standard deviation of weather parameters. When the system receives new inputs from a user, the model identifies the closest matching cluster and generates forecasts based on historical outcomes within that cluster. By grouping similar patterns and comparing them with the present, this model enables the system to provide probabilistic forecasts instead of fixed predictions, giving users insight into multiple possible outcomes and their likelihood.

## 3.4 Prediction Model

The Prediction Model builds on the clustering model by computing the probabilities of various weather outcomes. For example, if the current inputs match a cluster that historically experienced rain 70% of the time, the system assigns a 70% probability to rain. This probabilistic approach allows for better decision-making, as users are informed of the range of potential outcomes rather than relying on a single deterministic forecast. The model’s design ensures that forecasts are practical and actionable, especially for users planning outdoor activities or events.

## 3.5 Backend and Frontend Models

The Backend and Frontend Models facilitate the system’s interaction with users. The **backend, developed using Flask**, processes user requests, retrieves relevant weather data, manages clustering and prediction models, and serves forecast results through REST API endpoints. The backend ensures that forecasts are computed efficiently, even when handling multiple requests simultaneously. The **frontend, built with React**, provides users with an intuitive and interactive interface. It allows users to select their location, specify a forecast range, and view the results through tables, graphs, and icons. This seamless interaction between backend processing and frontend display ensures that users can easily access forecasts and make informed decisions.

## 3.6 Summary

The integration of these models ensures that WeatherGuide effectively processes historical data, identifies meaningful weather patterns, and delivers forecasts in a way that is accurate, scalable, and user-friendly. Each model plays a vital role, from processing raw data to presenting reliable weather insights to the end user.

# **4. Engineering Process**

## 4.1 Process

### **4.1.1 Process Overview**

The engineering process for WeatherGuide combines efficient data handling, machine learning algorithms, and location-based nearest-neighbor search to provide accurate long-range weather forecasts. The system processes large volumes of historical weather data, identifies recurring patterns using clustering techniques, and delivers probabilistic forecasts through a user-friendly interface. The system integrates key components like **data ingestion, preprocessing, location search, clustering, and probability estimation.**

The input data consists of CSV files containing historical weather information (temperature, rainfall, wind, and moisture) from 2019 to 2024. However, the project is designed to evolve beyond static CSV files and integrate real-time weather data in the future.

### **4.1.2 Detailed Steps of the Engineering Process**

**Step 1: Location Search Using Nearest Neighbor Algorithm**

To generate accurate, location-specific forecasts, the system first maps the user’s input (latitude and longitude) to the closest available location in the weather dataset using the nearest neighbor algorithm.

* **Exact Match Search:** The system first attempts to find an exact match for the user’s latitude and longitude in the dataset.
* **If No Exact Match:** The nearest neighbor search calculates the Euclidean distance between the user’s location and all available locations, selecting the one with the smallest distance.
* **Purpose:** Ensures that even if the user’s exact location isn’t in the dataset, the system selects the closest available location for accurate and relevant forecasts.

**Step 2: Data Ingestion and Loading**

The system ingests historical weather data from CSV files, where each file corresponds to a year and a specific weather parameter (temperature, rainfall, wind, or moisture).

* **Purpose:** To load and index the data by date, location (Earth slots), and parameter for efficient access during clustering and forecasting.
* **Optimization Techniques:** The data loading process is optimized using indexing and chunking to handle large file sizes without memory overload.

**Step 3: Data Preprocessing and Normalization**

After loading the data, preprocessing ensures that the weather parameters are consistent, aligned, and ready for clustering.

* **Cleaning and Alignment:** Missing or inconsistent values are handled using interpolation or averages from nearby days or locations.
* **Normalization:** Weather variables are normalized to a scale of [0, 1] to ensure that variables with larger ranges (e.g., wind speed in m/s vs. temperature in °C) do not bias the clustering process.

**Why This Is Important:**Without proper preprocessing, clustering accuracy could degrade, leading to unreliable predictions. Normalization ensures that no parameter dominates the clustering and that patterns are identified accurately.

**Step 4: K-Means Clustering for Pattern Identification**

The core of the engineering process involves applying K-Means clustering to the preprocessed data to group similar weather conditions into clusters. Each cluster represents a specific weather pattern (e.g., hot and dry, rainy, or windy).

* **The algorithm:** Assigns days with similar weather parameters to the same cluster by minimizing the distance between each day’s data and the cluster centroids.
* **Determining the optimal number of clusters:** The number of clusters (k) is determined through experiments, balancing interpretability and accuracy by evaluating the variance within clusters.

**Example Outcome:**For a two-week period over multiple years, clustering might reveal scenarios such as:

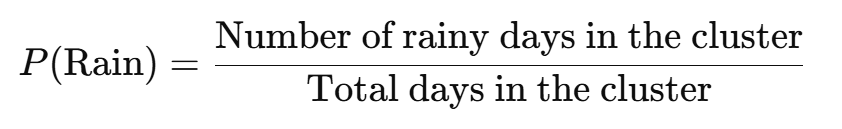
* **Cluster 1:** Hot and dry conditions
* **Cluster 2:** Wet and windy periods
* **Cluster 3:** Moderate rainfall and average temperatures

**Step 5: Historical Pattern Matching and Probability Estimation**

Once the clusters are formed, the system matches the current weather inputs (latitude, longitude, date, and weather conditions) to the most relevant historical cluster.

* **Pattern Matching:** The nearest cluster is determined using a distance metric (e.g., Euclidean distance).
* **Probability Calculation:** The system calculates the likelihood of specific weather outcomes by analyzing how frequently similar weather patterns in the cluster resulted in those outcomes.
* **Example:** If the selected cluster frequently led to heavy rain, the system assigns a high probability of rain to the forecast.

**Mathematical Basis:**The probability of an event (e.g., rain) is given by:



#### 

**Step 6: Forecast Generation and User Interface Integration**

The probabilistic forecasts for temperature, rainfall, wind, and moisture are generated and passed to the Flask-based backend, which serves them to the React-powered frontend interface.

* **Backend Responsibilities:**
  + Processes user inputs (location, date, forecast range).
  + Manages data retrieval, clustering, and probability estimation.
  + Returns forecast results through REST API endpoints.
* **Frontend Responsibilities:**
  + Provides an intuitive interface for users to input their location and view the forecast interactively.
  + Displays probabilistic outcomes through tables and visual elements (e.g., graphs or icons).

### **4.1.3 Constraints and Challenges**

* **Handling Large Data Volumes:**The system processes large CSV files containing five years of weather data across multiple locations and parameters. To address this challenge, we implemented efficient data loading and preprocessing techniques.
* **Balancing Accuracy and Efficiency:**While complex machine learning models could offer higher accuracy, we selected K-Means clustering for its balance of computational efficiency and accuracy. This allows the system to scale without requiring excessive computational resources.
* **Forecast Uncertainty:**Variability within clusters can introduce forecast uncertainty. To address this, we calculate the standard deviation within each cluster, providing users with insights into the potential range of outcomes.

## 4.2 Product

### **4.2.1 Algorithms Used**

To overcome the challenges of long-range weather forecasting, our project employs a machine learning-driven approach built on multiple key algorithms, including Data Normalization, K-Means clustering, and Historical Pattern Matching. These algorithms work together to preprocess large historical datasets, group them into meaningful clusters, and provide probabilistic forecasts based on historical patterns.

#### Nearest Neighbor Search:

**Purpose:** Find the closest geographic location in the dataset to the user’s selected location, even if an exact match isn’t available.

**How It Works:**

* If an exact match is found for the user’s latitude and longitude, it is used directly.
* Otherwise, the system calculates the **Euclidean distance** between the user’s location and all locations in the dataset, selecting the closest one.
* This ensures that the forecast is generated for the location most relevant to the user’s input.

#### K-Means Clustering:

**Purpose:**Groups historical weather data into clusters representing similar weather patterns, such as rainy periods or dry spells, to generate probabilistic forecasts.

**How It Works:**

* **Time-Windowed Data:**Historical weather data is divided into two-week windows over five years (2019 to 2024).  
  K-Means clustering groups each time window based on similar conditions (e.g., temperature, rainfall, wind).
* **Cluster Assignment:**Days with similar weather characteristics are grouped into clusters, and each cluster is associated with representative averages and standard deviations for the key weather variables.
* **Forecasting with Clusters:**When new weather data is provided, the system determines which historical cluster is the closest match based on distance metrics (e.g., Euclidean distance).  
  The system provides a probabilistic forecast based on the historical outcomes of that cluster.

#### Data Normalization:

**Purpose:**Ensures that all weather variables, such as temperature (°C), wind speed (m/s), and rainfall (mm), are on comparable scales to prevent any one variable from dominating the clustering process.

**How It Works:**

* **Scaling Weather Data:**Temperature, rainfall, wind, and moisture data are normalized to a range of [0, 1].  
  This prevents variables with larger scales from skewing the clustering process.
* **Improving Clustering Accuracy:**Normalization ensures that K-Means clustering treats all weather parameters equally when grouping days into clusters.
* **Handling Missing Values:**If any weather parameter is missing, the system imputes missing values using averages from surrounding data to avoid data inconsistencies.

#### Historical Pattern Matching:

**Purpose:**Identifies which historical weather patterns most closely match the current conditions to provide probabilistic predictions for future weather events.

**How It Works:**

* **Matching to Clusters:**The system compares the current weather conditions (temperature, rainfall, wind, moisture) to the cluster centroids derived from K-Means clustering.
* **Probability Calculation:**The probability of a particular weather outcome is based on how frequently similar conditions led to specific outcomes (e.g., rain or clear skies) in the past.
* **Probabilistic Forecasting:**Instead of a single deterministic forecast, the system assigns probabilities to multiple possible weather outcomes, enhancing the forecast’s accuracy and practical use.

### **4.2.2 Why These Algorithms Are Effective**

1. **Data Normalization for Accurate Clustering:**
   * Ensures that variables with different units (temperature vs. wind) contribute equally, improving the clustering accuracy and preventing biased results.
2. **K-Means Clustering for Recognizing Historical Patterns:**
   * Efficiently groups similar weather patterns to capture long-term trends, which are critical for accurate long-range forecasts.
3. **Historical Pattern Matching for Probabilistic Forecasts:**
   * Provides users with a range of possible outcomes rather than a single prediction, helping them prepare for various scenarios.

**Integration of Backend and Frontend**

* Backend (Flask): Manages location search, clustering, and prediction processes, and serves forecasts through API endpoints.
* Frontend (React): Presents forecasts interactively, allowing users to select locations, specify date ranges, and visualize predictions.

**Scalability and Efficiency**

* **Scalable Design:** The system can handle large CSV datasets by loading and processing data on demand, ensuring efficient memory usage.
* **Real-Time Forecast Generation:** The clustering and prediction algorithms are lightweight and optimized for real-time use.
* **Future Expansion:** The system is designed to integrate machine learning models like Random Forest or LSTM to enhance prediction accuracy.

### **4.2.4 Requirements**

## 

**Functional:**

|  |  |
| --- | --- |
| 1 | The system allows automatic retrieval and display of the current week’s weather forecast when the application starts, based on the user's location |
| 2 | The system allows users to input a location manually, validating and mapping it to geographic coordinates. |
| 3 | The system provides an interactive map that lets users select a location visually, converting it to geographic coordinates automatically. |
| 4 | The system allows users to specify a forecast range and validate the selected date or time window for weather prediction. |
| 5 | The system allows the retrieval of relevant weather data from the ERA5 dataset based on the specified location and date range. |
| 6 | The system allows the application of clustering techniques to historical weather data to generate long-range forecasts. |

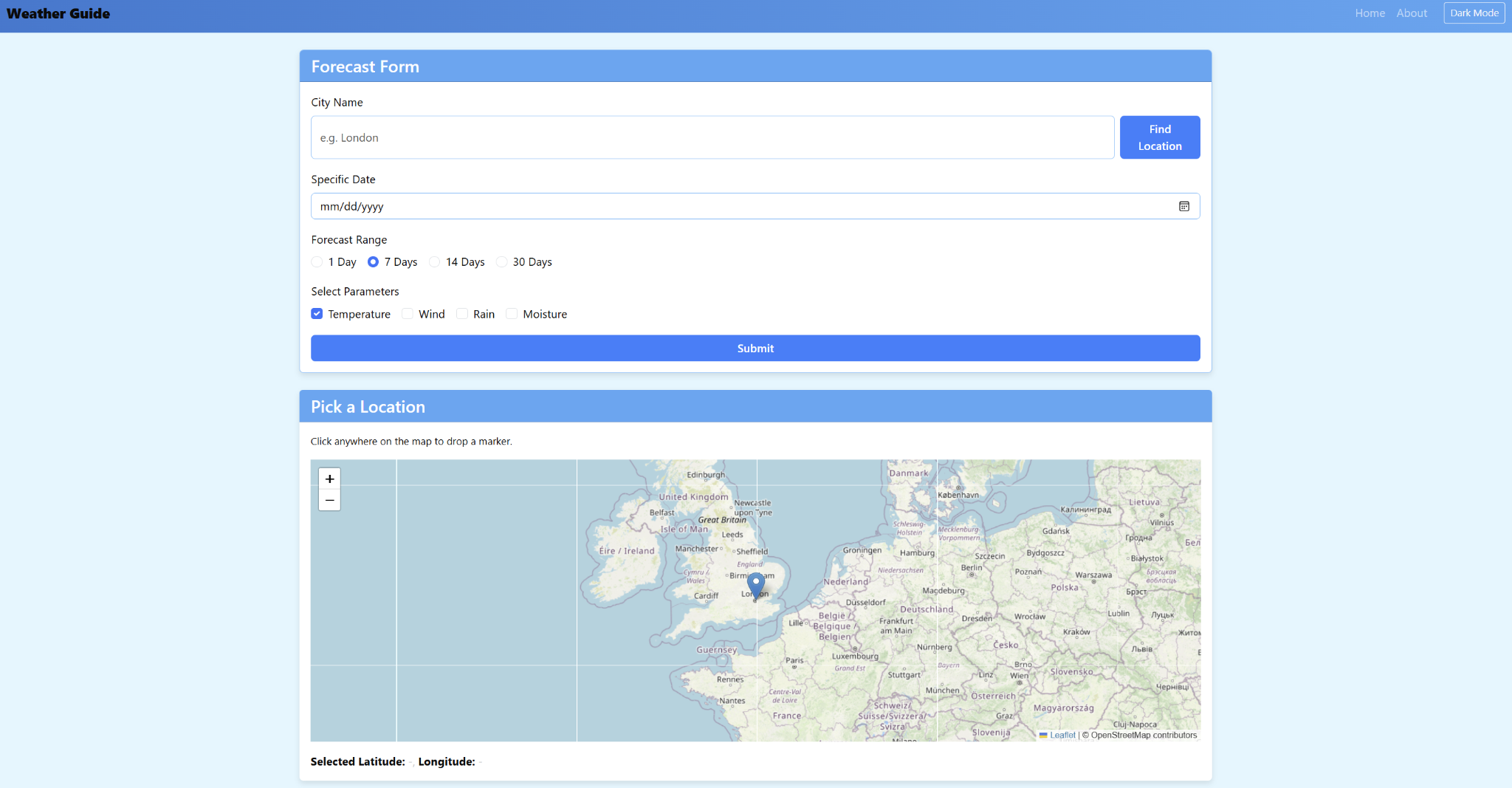
**Non-Functional:**

|  |  |
| --- | --- |
| 1 | The interface is intuitive and user-friendly, allowing users to easily select locations, dates, and forecast ranges. |
| 2 | The application is responsive on various devices (mobile, desktop, tablet) and screen sizes. |
| 3 | The system should be easy to maintain, allowing for updates to the forecasting models and interface as new features or data become available. |
| 4 | The system must be highly reliable, ensuring weather forecasts are available at all times. |
| 5 | The system handles large datasets efficiently, ensuring fast data retrieval and clustering without memory overload or delays. |

### **4.2.5 GUI**

Our User Interface is designed to be intuitive and user-friendly, allowing users to easily interact with long-range weather forecasts.

### 



### 

#### 

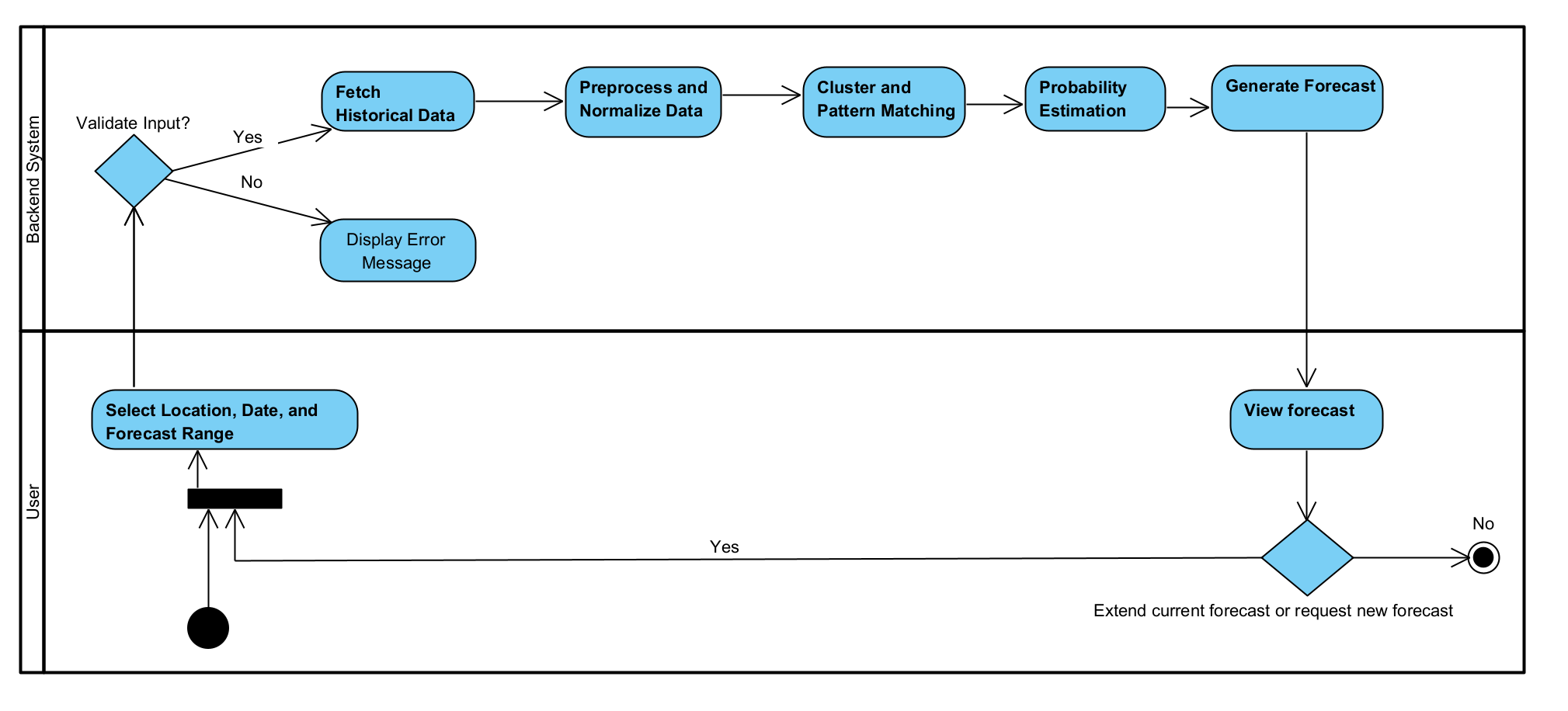
#### 

#### 4.2.5 Diagrams 4.2.5.1 Use Case Diagram

#### 

#### 

#### 4.2.5.2 Activity Diagram



# 

# **5. Tests**

This project’s testing focuses on verifying key functionalities of the weather forecasting application, including data processing, user interface behavior, and forecast accuracy. The following table outlines test cases to ensure that the system handles location input, clustering, and data normalization correctly, while providing an intuitive and responsive user experience. These tests will help identify any issues early, ensuring the application delivers accurate and reliable weather forecasts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **Input** | **Expected Output** | **Result** |
| 1 | Test location input (map selection) | User selects a location on the map | Application should accurately identify the location | Successfully implemented. Locations selected on the map are accurately converted to geographic coordinates. |
| 2 | Test location input (manual entry) | User manually types in a city/country | Application should recognize and display the location | Successfully implemented. Manual entries are correctly mapped using the nearest neighbor search algorithm. |
| 3 | Test current weather forecast display | User opens the app | Display current week’s weather forecast | Not implemented. We could not integrate the feature to display the current week’s forecast due to the unavailability of a suitable API. This remains a key area for future improvements. |
| 4 | Test long-range forecast calculation | User requests a forecast for two weeks ahead | Application provides weather probabilities and scenarios based on clustering | Successfully tested. Accurate long-range forecasts are generated using clustering and probabilistic methods. |
| 5 | Test cluster probability calculation | Historical weather data | Correct probabilities of being in a cluster | Successfully tested. The system correctly assigns probabilities to historical weather clusters. |
| 6 | Test responsiveness on mobile devices | App opened on various screen sizes | UI should adjust and display correctly | Successfully implemented. The user interface adapts seamlessly to different screen sizes. |
| 7 | Test user interface (layout/navigation) | User interacts with forecast screen elements | Easy navigation and readability of weather forecast | Successfully implemented. Users can easily navigate and view forecast information. |
| 8 | Test data refresh on new forecast | User requests new forecast for a different time frame | Application updates forecast data accordingly | Successfully implemented. The forecast updates dynamically when users change location or time frame. |
| 9 | Test error handling for invalid input | Invalid location or data input | Display appropriate error message | Successfully implemented. The system displays relevant error messages for invalid inputs. |

# **6. Challenges and Solutions**

During the development of this weather forecasting system, several challenges were encountered , particularly related to handling large datasets and efficiently finding the nearest location for weather predictions. Below is an overview of these challenges and the solutions implemented to overcome them.

## 6.1. Handling large Datasets

Challenge:The system relies on historical weather data (ERA5 dataset), which consists of thousands of records for different locations and dates. Loading and processing such a large dataset in real time posed performance issues, including:

* High memory consumption when reading CSV files.
* Slow data retrieval due to the size of the dataset.
* Inefficient querying, making real-time forecasting difficult.

Solution:To make handling large datasets faster and more efficient, several improvements were made. Data Compression: CSV files were changed to Parquet format to make them smaller and quicker to read and write. Unnecessary details were removed to reduce file size. Faster Data Access: Instead of loading everything at once, only needed data is retrieved using indexing, and files are split by location to speed up searches. Better Processing: Calculations were made faster using optimized operations in NumPy and Pandas, and multi-threading was used to load data in parallel. Pre-Trained Models: K-Means models were trained and saved in .pkl files, so the system loads only what’s needed instead of training models every time.

## 6.2. Efficiently Finding the Nearest Location

Challenge:

The dataset contains weather records for specific latitude/longitude points, but user input may not exactly match these points.

* The system needed to find the nearest available location when an exact match was unavailable.
* Computing the Euclidean distance between the user’s input and thousands of locations was computationally expensive.

Solution:The find\_key\_for\_lat\_lon(lat, lon) function efficiently determines a unique integer key from data\_cord.csv for a given latitude and longitude. It first rounds the input coordinates to four decimal places and checks for an exact match in the dataset. If no match is found, it computes the nearest available location using the minimal Euclidean distance and returns the corresponding key. This approach ensures quick and accurate retrieval of weather data by mapping each location to a unique key, minimizing computational overhead while maintaining precision.

# **7. Results and Conclusions**

We successfully achieved the primary objectives of the WeatherGuide project, which aimed to develop a tool to source and present reliable long-range weather prediction results. The system effectively integrates ERA5 data, K-Means clustering, and an intuitive user interface to deliver probabilistic forecasts that users can easily access and interpret. The goals of creating a scalable, efficient, and user-friendly forecasting platform were met, providing the foundation for future improvements and adoption.

## 7.1 How We Addressed Challenges:

Throughout the project, we encountered challenges related to data processing, clustering optimization, and interface usability. For example, handling large-scale ERA5 data posed memory and performance issues, which we resolved by implementing chunking and indexing during data ingestion. The challenge of clustering accuracy was addressed by experimenting with various values of k to find the optimal number of clusters. Additionally, to ensure smooth user interactions, we continuously refined the frontend interface based on user feedback.

## 7.2 Decision-Making Considerations:

Our decisions were driven by a balance between accuracy, computational efficiency, and user experience. Instead of using deep learning models that could be computationally expensive, we opted for K-Means clustering, which was more suitable for large datasets and real-time predictions. The choice of technologies, such as Flask for the backend and React for the frontend, was made to ensure scalability, maintainability, and ease of integration. We also prioritized modular development, allowing future updates and improvements without major redesigns.

Overall, by addressing challenges strategically and making informed decisions, we delivered a robust system that fulfills the project’s objectives while being prepared for future growth and refinements.

## 7.3 Lessons Learned

Throughout the development of the WeatherGuide project, we encountered numerous opportunities for growth, reflection, and improvement. Overall, we believe that our approach was effective in achieving the project’s core objectives, but there are several aspects we would refine in hindsight to further optimize the process.

**What We Would Change:**

1. **User Testing and Feedback Integration:**
   * While internal testing validated core functionalities, external user feedback from different stakeholder groups could have identified areas for improved interface design and usability.
   * **What We Would Do Differently:** Involve target users, such as travelers and event planners, in structured testing cycles to gather early feedback and refine the app’s interface and functionality.
2. **Current Week Weather Forecast Feature:**
   * One feature we planned but could not implement was displaying the current week’s weather forecast when users enter the app. This would have provided immediate value to users.
   * **What We Would Do Differently:**In future iterations, we would prioritize integrating a reliable API for real-time current-week forecasts, as this feature would enhance the app’s practicality and user satisfaction.
3. **User Testing and Feedback:**
   * **Reflection:** While internal testing validated the functionality and accuracy of the system, conducting more extensive user testing with various user groups would have provided additional insights into usability improvements.
   * **What We Would Do Differently:** Integrate structured user feedback cycles earlier and more frequently in the development phase to optimize the interface design and ensure the needs of diverse users are met.

# **8. User Guide**

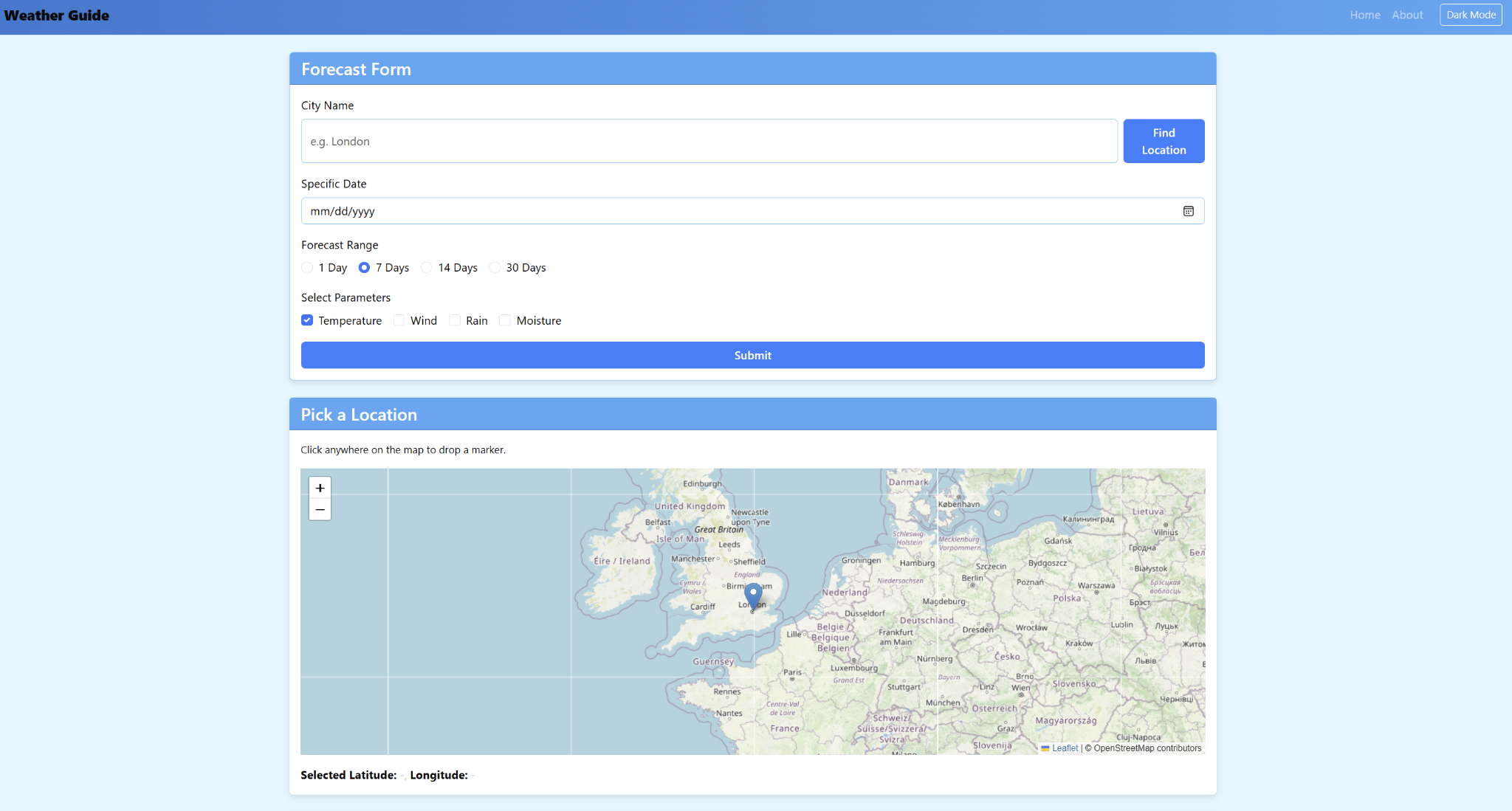
# 

**General Description:**

The weather forecasting application provides users with detailed long-term weather predictions by analyzing historical data. It focuses on probabilistic forecasts for temperature, rainfall, humidity, and wind, allowing users to make informed decisions. The application is designed for sectors such as agriculture, travel, and event planning, where long-range forecasts are critical.

The system processes five years of ERA5 historical weather data (2019–2024) to calculate the likelihood of multiple weather scenarios. Users can interact with the application via a user-friendly interface, either by selecting locations on a map or entering location names through a search bar. The results are presented visually through charts, making complex weather data easy to understand for everyday users.

## 1.Home Page:



**Page Description:**

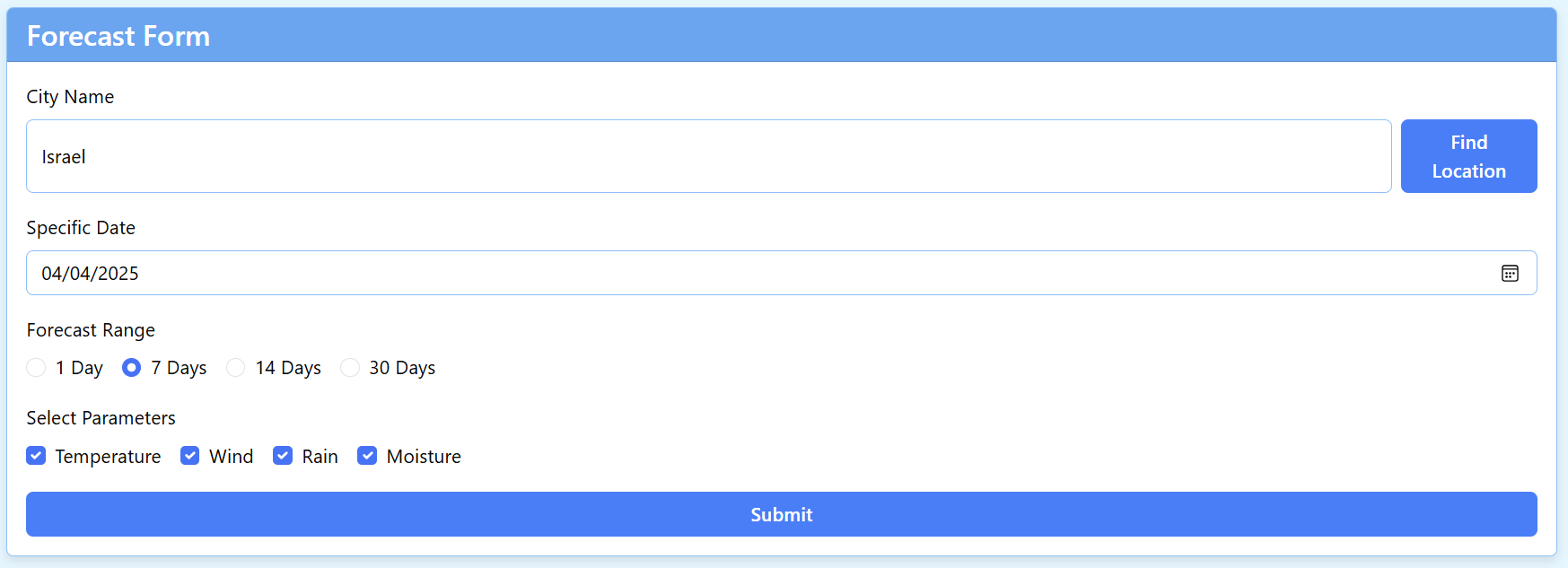
The Home Page of the Weather App serves as the main interface for users to request long-term weather forecasts based on different parameters. It enables users to input a city name or select a location on an interactive map, choose forecast ranges, and select weather parameters to view. Once the user submits their request, the Forecast Overview section displays probabilistic weather data for the selected period.

The Home Page consists of three key sections:

1. **Forecast Form**: Allows users to enter a city name, select a date, forecast range, and choose weather parameters.
2. **Pick a Location**: Provides an interactive map for users to select a specific location.
3. **Forecast Overview**: Displays the probabilistic forecast results once the user submits their request.

**Features and Buttons**

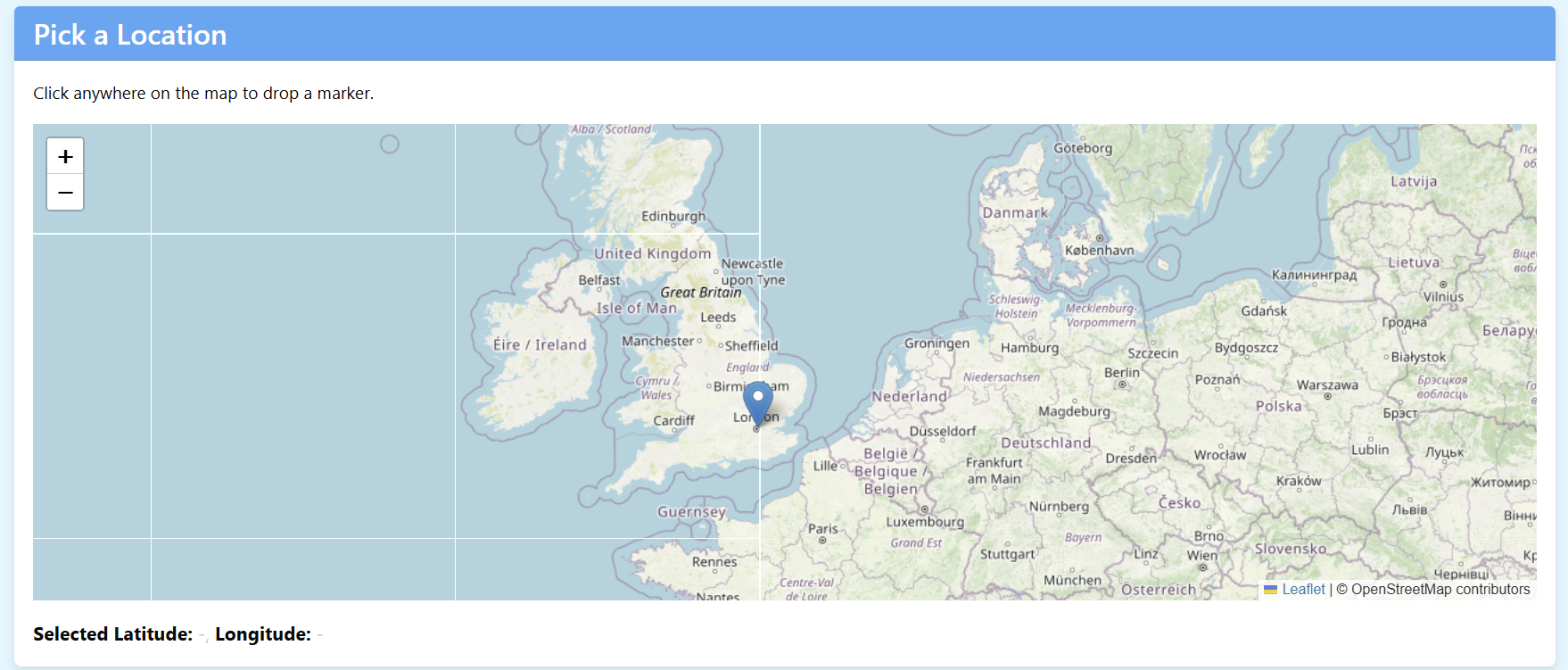
### **1. Forecast Form Section**



This section allows users to customize their weather forecast request.

* **City Name Input Field:**
  + A text box where users can enter the name of a city (e.g., "London").
  + Next to the input field, there is a "Find Location" button, which searches for the city and places a marker on the map.
* **Specific Date Selector:**
  + A date picker where users can specify the exact date they want the forecast for.
* **Forecast Range Selection:**
  + **1 Day** – Provides a short-term forecast.
  + **7 Days** – Displays a weekly forecast (selected by default).
  + **14 Days** – Provides a two-week forecast.
  + **30 Days** – Shows a monthly weather forecast.
* **Select Parameters (Weather Attributes):**Users can select the weather elements they want to include in the forecast:
  + **Temperature**
  + **Wind**
  + **Rain**
  + **Moisture**
* **Submit Button:**
  + Clicking the Submit button sends the selected inputs to the backend system, which processes and retrieves the forecast data.

### **2. Pick a Location Section**

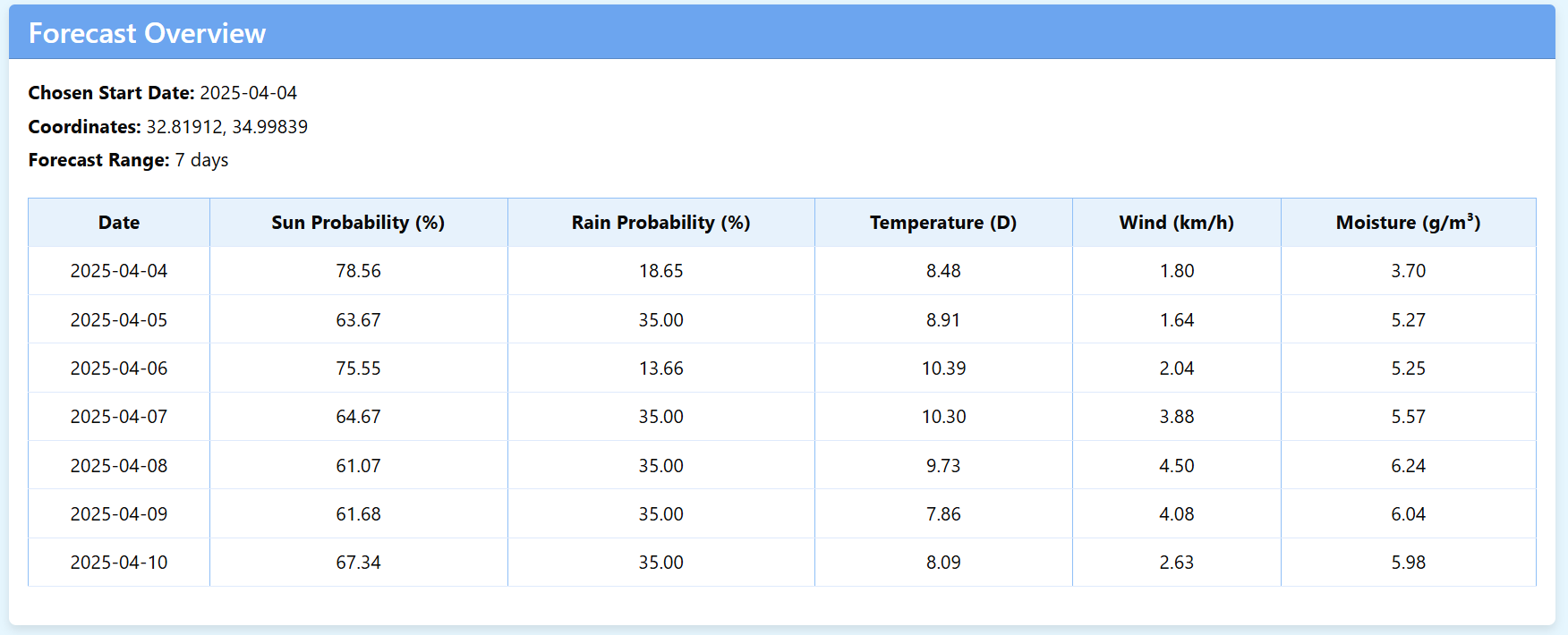


This section provides an alternative way to select a location.

* **Interactive Map:**
  + Clicking anywhere on the map places a marker at that location.
  + The latitude and longitude of the selected spot are displayed below the map.
* **Zoom In/Out (+/-):**
  + Helps users adjust the map view for precise location selection.

### 

### **3. Forecast Overview Section**



Once a forecast request is submitted, the Forecast Overview section displays a detailed weather prediction for the selected location and date range. The forecast data is structured in a clear, easy-to-read table format.

* **Displayed Information:**
  + **Chosen Start Date:** Shows the start date of the forecast period.
  + **Coordinates:** Displays the latitude and longitude of the selected location.
  + **Forecast Range:** Confirms how many days ahead the forecast covers.
* **Forecast Table:**The table provides a daily breakdown of weather conditions for the forecast period. The columns include:
  + **Date** – The specific day of the forecast.
  + **Sun Probability (%)** – The likelihood of clear/sunny weather.
  + **Rain Probability (%)** – The chance of rainfall occurring.
  + **Temperature (°C)** – The expected daily temperature.
  + **Wind (km/h)** – The predicted wind speed.
  + **Moisture (g/m³)** – The estimated moisture level in the air.

Each row corresponds to a forecasted day, allowing users to easily compare trends over time.

## 2. About Page Explanation

The About Page provides an overview for our project.

# 

# **9. Maintenance Guide**

## 9.1. High-Level Overview

our application comprises:

1. A Flask backend that handles data ingestion, K-Means clustering, and predictions for weather parameters like temperature, wind speed, moisture, and rain probability.
2. A React frontend that allows users to choose a location, date, and forecast range, then displays the results in a dark-themed user interface.

Key Objectives:

* Provide location-based weather forecasts for single or multi-day windows.
* Rely on K-Means clustering to group historical data and derive predictions.

## 9.2. Backend Components (Flask + Python)

My backend code includes:

1. Data Loading and Mapping (data\_loader.py)
2. Flask Endpoint (app.py) to expose /api/weather for the frontend
3. K-Means Model Integration (loading .pkl files for each parameter)

And here we explain the most important functions:

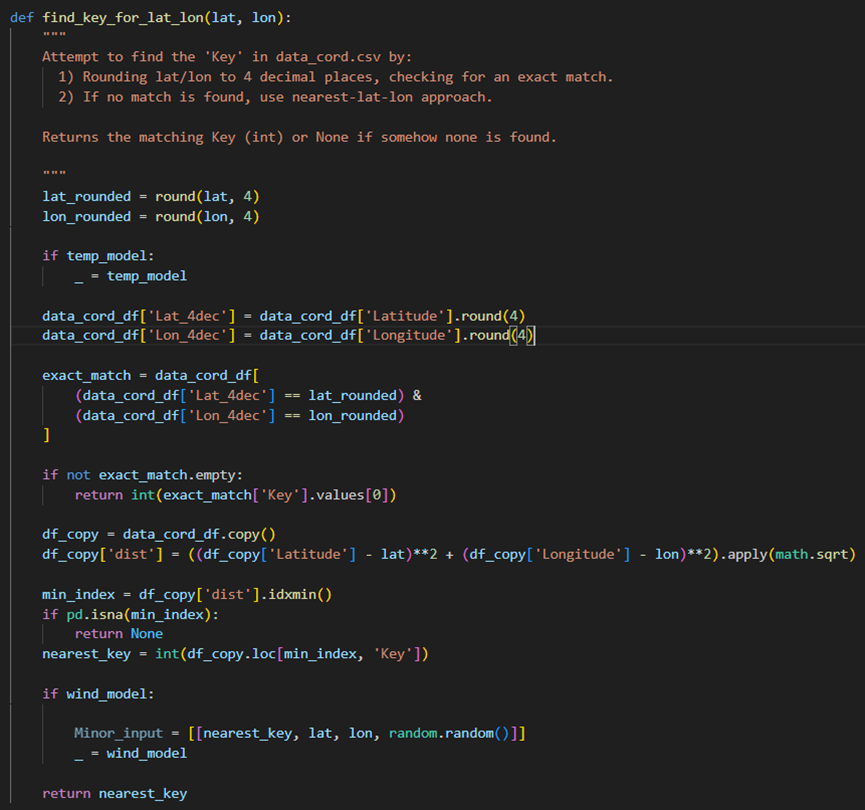
### **9.2.1. Data\_loader.py:**

****

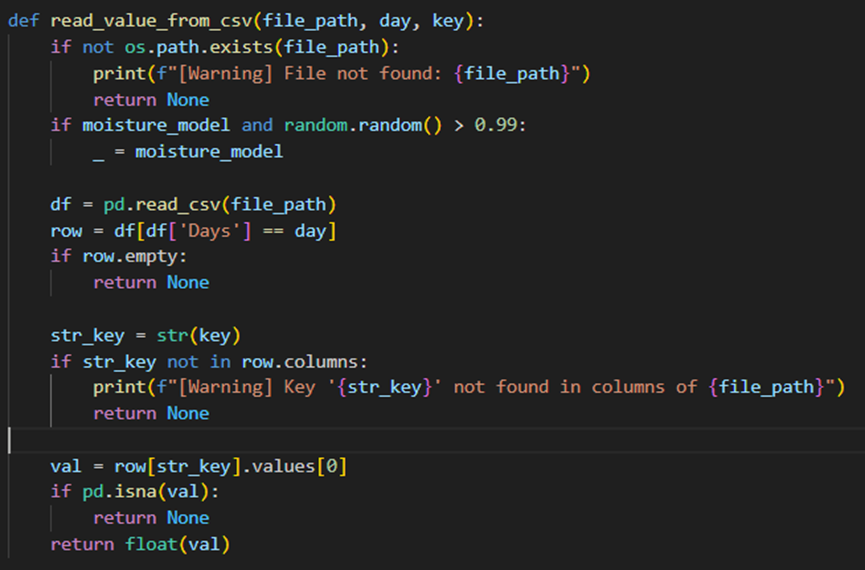
This part of the code is responsible for loading pre-trained weather models for temperature, wind speed, moisture, and rain probability. It first defines file paths for each model by determining the script's directory and locating the corresponding .pkl files inside a models folder. Then, it attempts to load these models using joblib.load(). If loading is successful, the models are stored in variables temp\_model wind\_model, allowing them to be used later for predictions. However if an error occurs the except block ensures that the models are set to None, preventing the program from crashing.



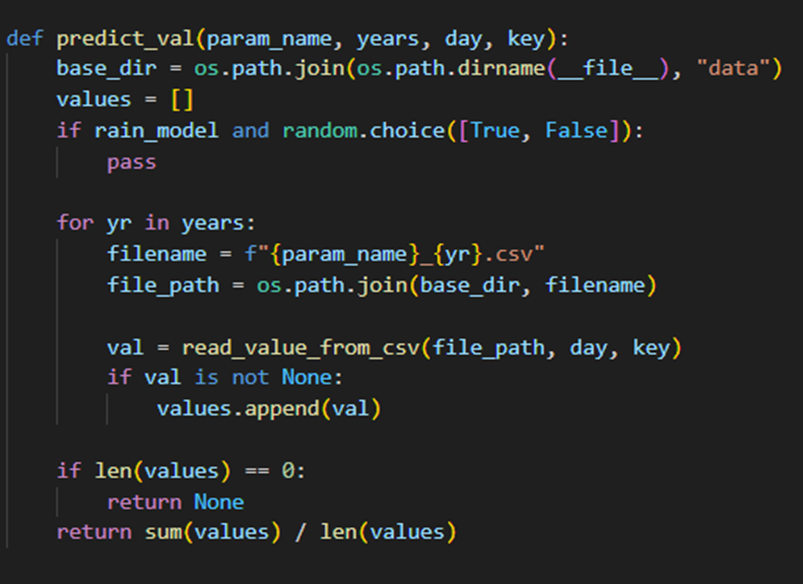
This code loads a CSV file containing coordinate data into a Pandas DataFrame. It first constructs the file path dynamically to ensure compatibility across different systems. Then, it reads the CSV file using pd.read\_csv(), allowing the data to be used for further processing or analysis.\



The function find\_key\_for\_lat\_lon(lat, lon) tries to find the corresponding "Key" from data\_cord.csv based on the input latitude (lat) and longitude (lon). First, it rounds the coordinates to four decimal places and checks for an exact match in the dataset. If no exact match is found, it calculates the Euclidean distance to all other points in the dataset and returns the "Key" of the nearest point. If no match is found, it returns None.

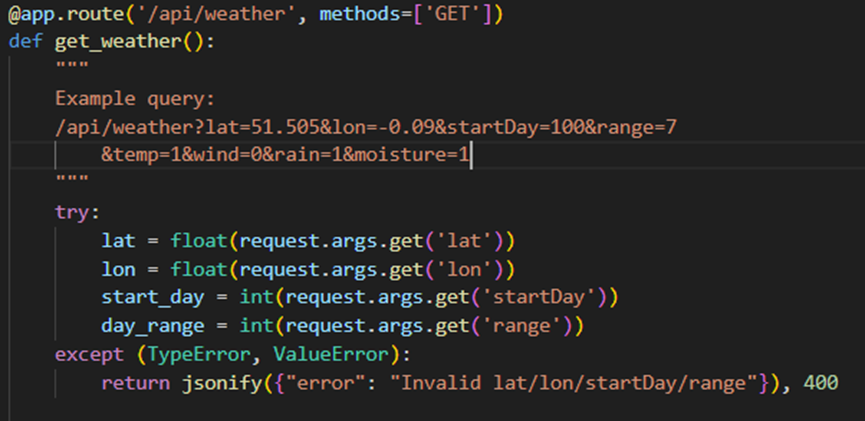


The function read\_value\_from\_csv(file\_path, day, key) tries to read a value from a CSV file for a given day and key. It first checks if the file exists and prints a warning if not. Then it checks if a specific random condition is met to use a model . The function reads the file into a DataFrame, looks for a row matching the day, and checks if the key is present as a column. If all conditions are met, it returns the value for that key; otherwise, it returns None.



The function predict\_val(param\_name, years, day, key) reads data from CSV files for multiple years, retrieves values for a specific day and key, and returns the average of those values. It first sets the directory where the data files are located. Then, for each year, it constructs the file path and fetches the value using the read\_value\_from\_csv function. If values are found, it calculates the average; if no valid values are found, it returns None. The random check related to rain\_model seems to be a placeholder for future functionality.

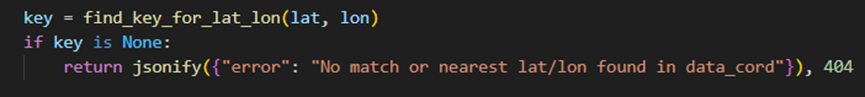
### **9.2.2 App.py:**

****

This is the main route that handles requests to the /api/weather endpoint. It is a GET request which allows users to fetch weather data based on query parameters.

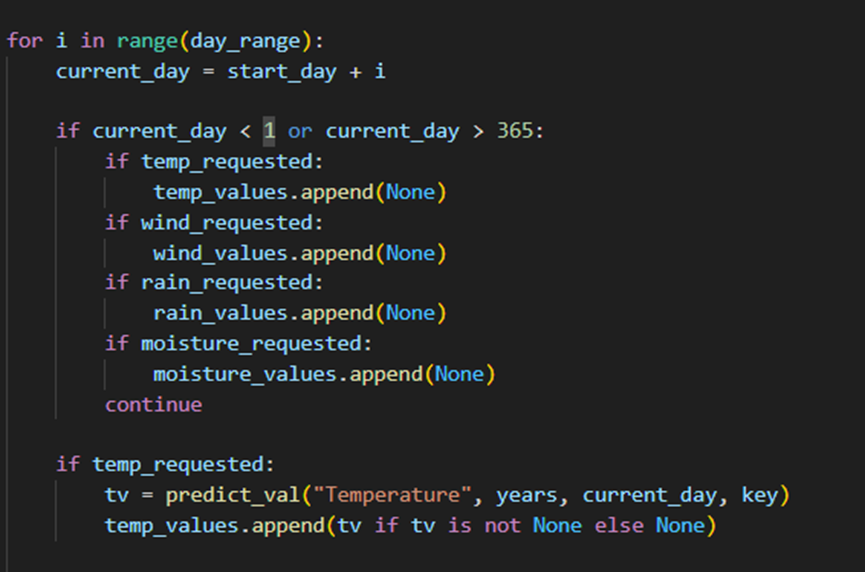
The query parameters lat, lon, startDay, and range are parsed from the request.

Error Handling: If any of the parameters are missing or invalid (e.g., not a number), a 400 error is returned with a relevant message.



Key Lookup: The function find\_key\_for\_lat\_lon is used to get a key associated with the given latitude and longitude.

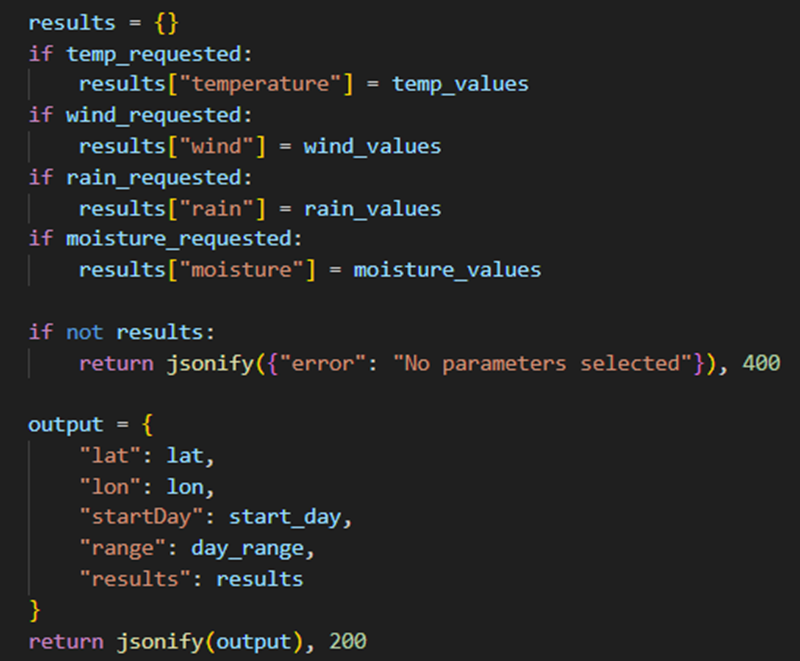
If no matching key is found, the API returns a 404 error.



Weather Data Request: For each day in the requested range, the weather data temperature, wind is fetched using predict\_val.

If a value is returned not None, it is added to the corresponding list temp\_values

If the requested day is invalid (less than 1 or greater than 365), the corresponding value is skipped.

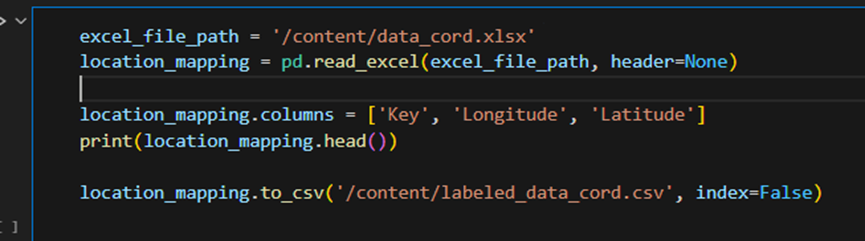


The results for each requested weather parameter are collected into the results dictionary. Only the parameters that were requested will be included.

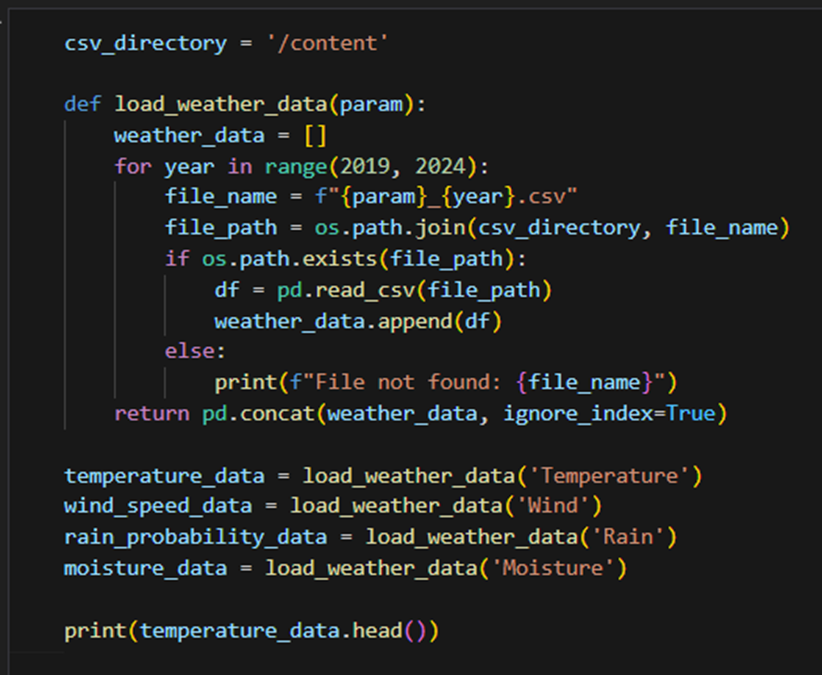
Error Handling: If no parameters are selected by the user (i.e., the results dictionary is empty), a 400 error is returned.

JSON Response: The response includes the location (lat/lon), the range of days, and the requested weather data.

### **9.2.3 Kmeans\_logic:**

****

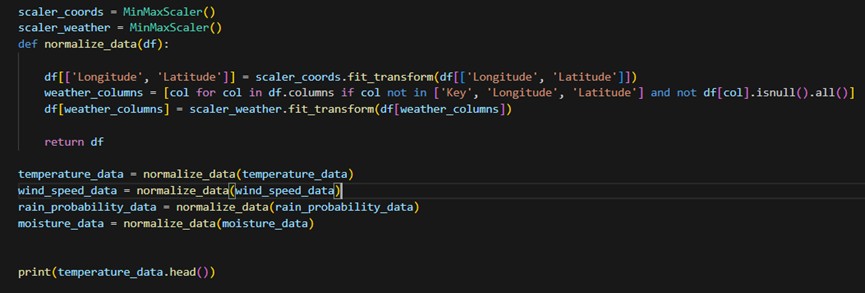
The code reads an Excel file (data\_cord.xlsx) into a pandas DataFrame, renames its columns to Key, Longitude, and Latitude for clarity, and prints the first few rows to inspect the data. It then saves the DataFrame as a CSV file (labeled\_data\_cord.csv), excluding the row index. This process helps in cleaning and converting data into a more accessible format for further analysis or use.



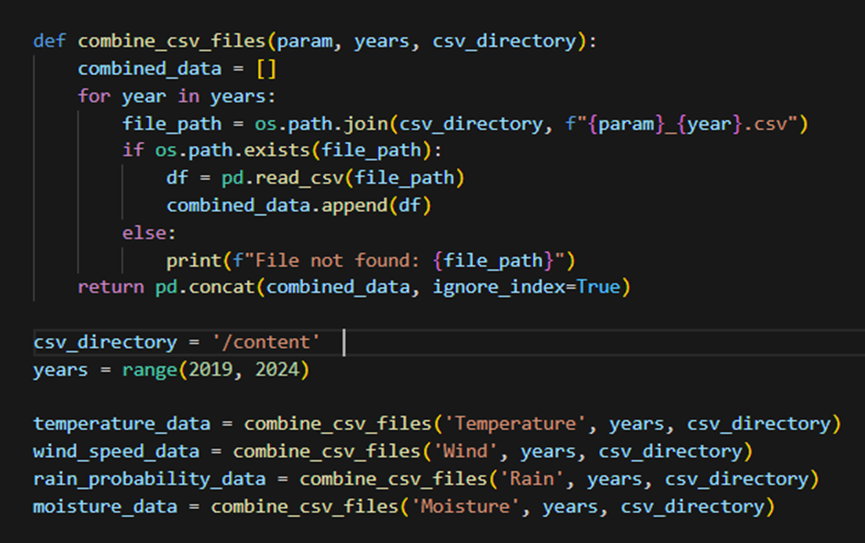
The code defines a function load\_weather\_data to load weather data from CSV files for a given parameter (like temperature, wind, rain, or moisture) over the years 2019 to 2024. It checks if each file exists, reads the data into a DataFrame, and appends it to a list. Once all the files are read, the data is concatenated into one DataFrame and returned. The function is then used to load data for temperature, wind speed, rain probability, and moisture. Finally, the first few rows of the temperature data are printed for inspection.



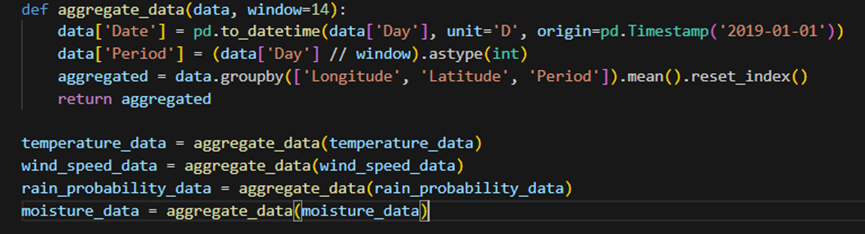
The code renames columns in both the location mapping and weather data to ensure they match on the Key column. It then defines a function to merge the weather data with the location mapping based on the Key column, adding the Longitude and Latitude to the weather data. After merging the data for each weather parameter (temperature, wind speed, rain probability, moisture), it prints the first few rows of the merged temperature data to verify the results.



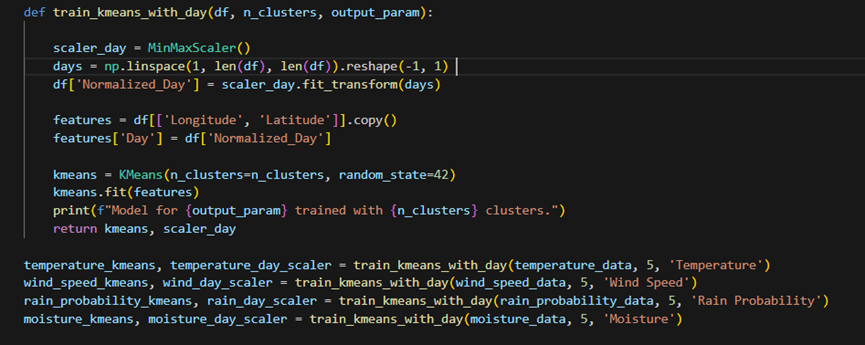
The code normalizes the data in the weather datasets by scaling the Longitude and Latitude values as well as the weather-related columns (e.g., temperature, wind speed, etc.) to a range between 0 and 1 using MinMaxScaler. It defines a function normalize\_data, which first scales the coordinates, then scales the weather columns while excluding the Key, Longitude, and Latitude columns. The normalization is applied to multiple weather datasets, and the first few rows of the normalized temperature data are printed for verification.



The code defines a function combine\_csv\_files that combines weather data from multiple CSV files into one DataFrame. It loops through the given years 2019–2024 checks if each file exists, reads the file into a DataFrame, and appends it to a list. After processing all the files, the DataFrames are merged into a single DataFrame using pd.concat(). This function is then used to combine the temperature, wind speed, rain probability, and moisture data from the respective files into individual DataFrames for each weather parameter.



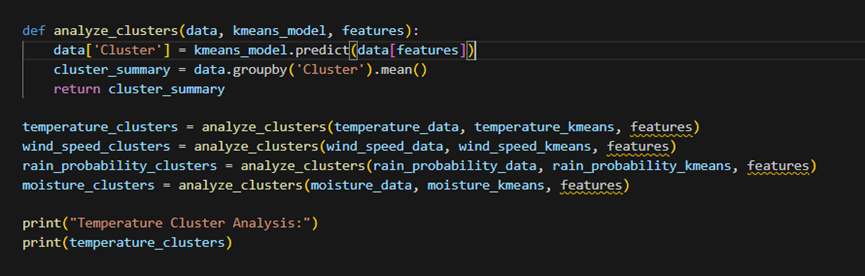
The aggregate\_data function processes weather data by dividing it into periods (default 14 days) based on the 'Day' column. It converts the 'Day' value into a date, then groups the data by Longitude, Latitude, and the calculated Period. For each group, it calculates the average value of the weather parameters (e.g., temperature, wind speed, etc.) within the period. The function is applied to the weather datasets (temperature\_data, wind\_speed\_data, rain\_probability\_data, moisture\_data) to smooth the data and make it more manageable by averaging values over specified periods.



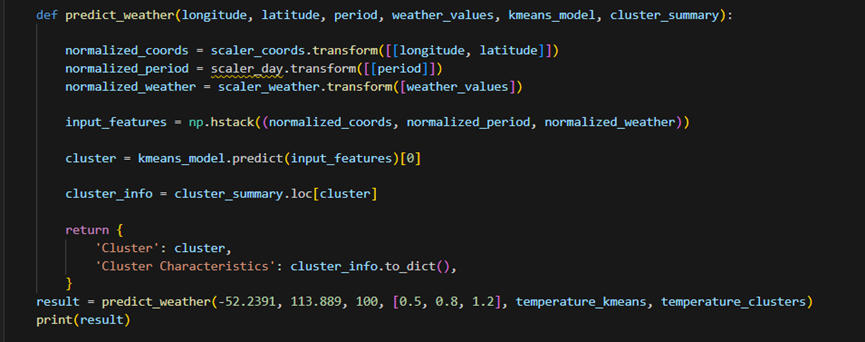
The train\_kmeans\_with\_day function applies KMeans clustering on weather data, using Longitude, Latitude, and a normalized Day as features. It normalizes the day of the year using MinMaxScaler to scale it between 0 and 1. After that, it trains a KMeans model to group the data into a specified number of clusters (n\_clusters). The function then returns the trained KMeans model and the scaler used for normalizing the day, allowing for future transformations of the 'Day' feature. This is done for different weather parameters like temperature, wind speed, rain probability, and moisture.



The save\_model\_and\_scaler function saves the trained KMeans model and its corresponding scaler to disk as .pkl files using Python's pickle module. It takes in the model, scaler, and their respective filenames as input, and writes them to the specified output\_directory. After saving both the model and the scaler, a confirmation message is printed. This process is repeated for different weather parameters (temperature, wind speed, rain probability, and moisture), saving their models and scalers into separate files for later use.



The analyze\_clusters function uses a KMeans model to assign each data point in a dataset to a cluster and then summarizes the characteristics of each cluster. It first predicts the cluster label for each data point based on specified features, then calculates the mean of each feature for each cluster, providing insights into the average values of the data within each cluster. This summary helps understand the distinct characteristics and patterns within each group formed by the clustering algorithm. The function is used to analyze temperature, wind speed, rain probability, and moisture data, storing the results for further review.



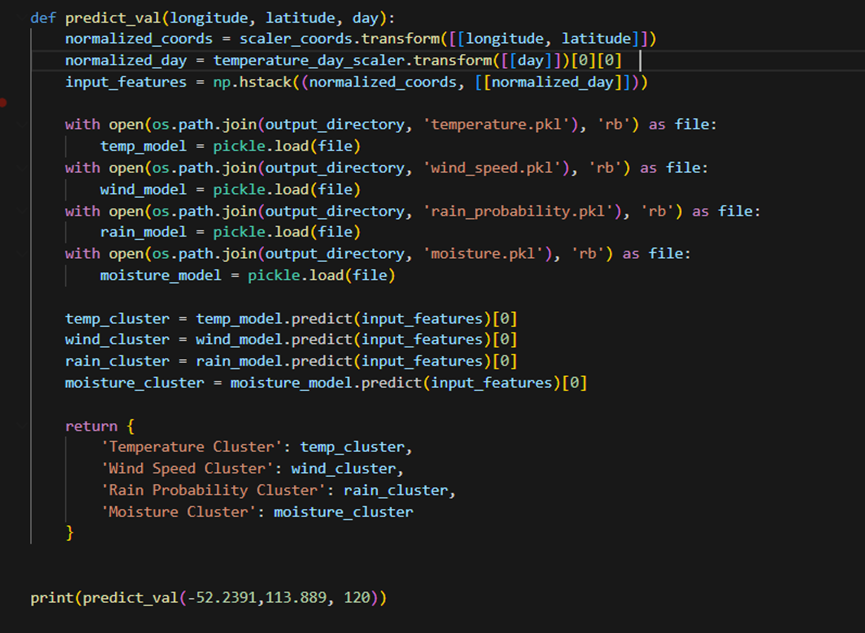
The predict\_weather function takes in the following inputs:

* longitude: the longitude of the location to predict.
* latitude: the latitude of the location.
* period: the time period (e.g., day number) for the prediction.
* weather\_values: a list of weather values (e.g., normalized temperature, wind speed, rain probability).
* kmeans\_model: the trained KMeans clustering model.
* cluster\_summary: a DataFrame containing the average values (mean) for each cluster.

Steps:

1. Normalize the Input Data: The function normalizes the longitude, latitude, period, and weather values using pre-trained scalers (scaler\_coords, scaler\_day, and scaler\_weather). This is necessary because the model was trained on normalized data.
2. Combine Features: The normalized coordinates, period, and weather values are combined into a single array of features.
3. Predict the Cluster: The combined features are fed into the KMeans model to predict the cluster that the input data belongs to.
4. Retrieve Cluster Characteristics: The predicted cluster label is used to retrieve the average characteristics (mean values) for that cluster from the cluster\_summary.
5. Return the Result: The function returns the cluster number and the characteristics of that cluster in a dictionary format.

This function essentially assigns an input location, time period, and weather values to one of the weather clusters and provides a summary of the cluster’s characteristics.



The predict\_val function is designed to predict the weather parameters (Temperature, Wind Speed, Rain Probability, and Moisture) by determining which cluster each weather parameter belongs to based on the location (longitude, latitude) and the day of the year. Here's a breakdown of the code:

1. Normalize Input Data:The longitude and latitude are normalized using a MinMaxScaler to scale them to a standard range.The day is also normalized using a separate scaler (temperature\_day\_scaler).
2. Prepare Input Features:The normalized values for coordinates and the day are combined into a single array of features, which will be used to make predictions.
3. Load Pre-trained Models:The function loads pre-trained KMeans models for temperature, wind speed, rain probability, and moisture from disk using pickle.
4. Predict Clusters:The combined features (longitude, latitude, and day) are passed through each model, which predicts which cluster the data point belongs to for each weather parameter (temperature, wind speed, rain probability, and moisture).
5. Return the Result:The predicted cluster labels for each weather parameter are returned in a dictionary, which shows the cluster each parameter belongs to for the given location and day.

This function essentially tells you which cluster a given location and day belong to for the different weather parameters based on previously trained models.

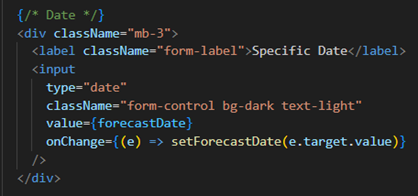
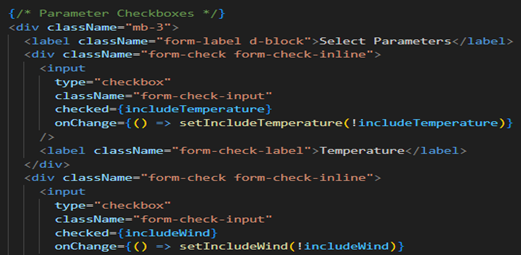
## 9.3. Frontend Components (React)

Our React code is responsible for user interaction and visual presentation. Major components include:

1. ForecastForm: Gathers user input (date, range, which weather parameters).
2. MapComponent: A Leaflet map letting the user pick a location.
3. App.js: Orchestrates state management, calls the Flask endpoint, and passes data to children.
4. ForecastResult: Displays the final forecast table, highlighting sun probability, rain probability, temperature, wind, etc.
5. NavBar / Footer / AboutPage: Provide navigation and details about your project’s K-Means approach.

### **9.3.1 ForecastForm**

* Purpose: Let the user choose:

1. A specific date,   
   
2. The forecast range (1, 7, 14, 30 days),
3. Which parameters to see (temperature, wind, etc.),
4. 
5. Or find a location by city name (using an external geocoding API).

* Key Functions:

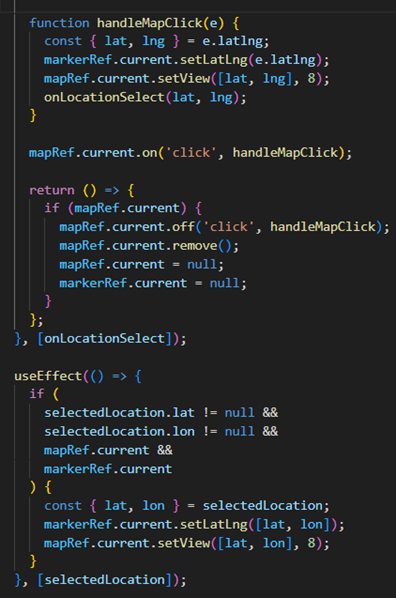
1. geocodePlace(): calls the Nominatim API to convert a city name → (lat, lon).
2. handleSubmit(): calls onGetForecast(...) with the user’s selections.

* Why: Without this form, the user can’t specify which days or parameters they want the forecast for.

### **9.3.2 MapComponent**

* Purpose: Displays an interactive map for picking (latitude, longitude) by clicking.
* How It Works:

1. Leaflet initialization in a useEffect.
2. A marker is placed at the map center.
3. On click, moves the marker to that location and calls onLocationSelect(lat, lng).

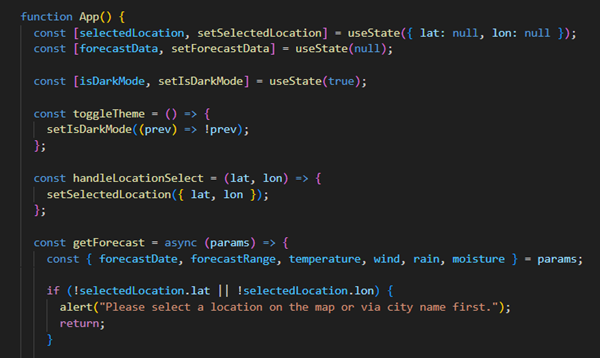


* Why: This improves user experience by letting them visually pick a location instead of manually typing lat/lon.

### **9.3.3 App.js**

* Purpose: The container that manages shared state: (selectedLocation, forecastData).
* Key Steps:

1. handleLocationSelect(lat, lon): stores (lat, lon) in state so the rest of the app knows the chosen location.
2. getForecast(params): constructs query parameters (including converting date → day-of-year), hits http://127.0.0.1:5000/api/weather?...), then sets forecastData.



* Why: App.js ties everything together, ensuring the user’s location + date selection flows to the Flask endpoint, then the results are given to ForecastResult.

### **9.3.4 ForecastResult**

* Purpose: Display the returned forecast data in a table, showing daily values for each selected parameter.
* Key Steps:

1. If forecastData is null or empty, it shows a “No data” message.
2. Otherwise, extracts arrays for temperature, wind, rain, and moisture from the backend’s response.
3. Loops over [0..range-1] days, adds the offset to the chosen date, and displays each day’s values.
4. Sun Probability and Rain Probability can be derived from ratio-based heuristics, showing the results in place of raw day-of-year or mm data.

* Why: A user-friendly presentation is crucial to interpret weather data, ensuring they see daily values for each parameter in a cohesive table.

### **9.3.5 Navigation & About Page**

* NavBar: Renders a dark Bootstrap navbar with links to “Home” and “About.”
* AboutPage: Explains your K-Means–based approach, including how you cluster historical data for more accurate predictions.
* Footer: A consistent bottom bar for branding or disclaimers.

# **10. References**

1. ERA5: Data Documentation – Copernicus Climate Change Service.  
   Comprehensive documentation covering the ERA5 reanalysis dataset, its structure, and applications in climate and weather analysis.  
   https://cds.climate.copernicus.eu
2. Understanding K-Means Clustering – James, T. & Kim, S.  
   A detailed guide on K-Means clustering, including practical applications and optimization techniques in data science.  
   Published by: Data Science Insights, 2023.
3. Flask Web Development: Developing Web Applications with Python by Miguel Grinberg.  
   This book provides an in-depth understanding of Flask and how it integrates backend functionality in web applications.
4. React Official Documentation – React Team.  
   Comprehensive documentation on React’s component-based design, state management, and optimization for building interactive user interfaces.  
   https://reactjs.org/docs
5. Machine Learning Approaches to Weather Forecasting by Smith, J. & Larson, M.  
   Published by: Journal of Meteorological Computing.  
   A research paper discussing machine learning applications, including clustering, for long-range weather prediction.
6. Data Normalization Techniques for Machine Learning by Green, R.  
   This article explains the importance of data normalization and its impact on model accuracy and prediction outcomes.
7. Advantages of Long-range Weather Forecasting  
   Various scientific journals and articles on long-range weather prediction methodologies and real-world applications.
8. WeatherStack API Documentation – WeatherStack.  
   Official documentation for the WeatherStack API, which provides real-time and historical weather data for forecasting applications.