AGRO WEATHER INFORMATION

Minor project-1 report submitted
in partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology
in
Artificial Intelligence & Machine Learning

By

R.K.GOWTHAM	(22UEAM0018)	**(VTU21354)**
G.DHEEPAK	(22UEAM0016)	**(VTU21577)**
M.VADHANA KUMAR	(22UEAM0063)	**(VTU23122)**

Under the guidance of
Dr.P.J.Beslin Pajila,M.E.,Ph.D .,
ASSISTANT PROFESSOR- -Senior Grade

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

october, 2024

AGRO WEATHER INFORMATION

Minor project-1 report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Artificial Intellegence & Machine Learning

By

R.K.GOWTHAM	(22UEAM0018)	**(VTU21354)**
G.DHEEPAK	(22UEAM0016)	**(VTU21577)**
M.VADHANA KUMAR	(22UEAM0063)	**(VTU23122)**

Under the guidance of
Dr.P.J.Beslin Pajila,M.E.,Ph.D .,
ASSISTANT PROFESSOR- -Senior Grade

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

october, 2024

CERTIFICATE

It is certified that the work contained in the project report titled ”AGRO WEATHER INFORMATION” by ”R.K.GOWTHAM (22UEAM0018), G.DHEEPAK (22UEAM0016),M.VADHANA KUMAR (22UEAM0063)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Dr.P.J.Beslin Pajila
Assistant Professor-Senior Grade
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
october, 2024

Signature of Head of the Department
Dr. S. Alex David
Professor & Head
Artificial Intellegence & Machine Learning
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
october, 2024

Signature of the Dean
Dr. S P. Chokkalingam
Professor & Dean
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
october, 2024

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)
R.K.GOWTHAM
Date: / /

(Signature)
G.DHEEPAK
Date: / /

(Signature)
M.VADHANA KUMAR
Date: / /

APPROVAL SHEET

This project report entitled AGRO WEATHER INFORMATION by R.K.GOWTHAM (22UEAM0018), G.DHEEPAK(22UEAM0016), M.VADHANA KUMAR (22UEAM0063) is approved for the degree of B.Tech in ARTIFICIAL INTELLIGENCE & MACHINE LEARNING.

Examiners **Supervisor**

Dr.P.J.Beslin Pajila,M.E.,Ph.D .,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our **Honorable Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (Electrical), B.E. (Mechanical), M.S (Automobile), D.Sc., and Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, for her blessings.

We express our sincere thanks to our respected Chairperson and Managing Trustee **Mrs. RANGARAJAN MAHALAKSHMI KISHoRE,B.E., Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, for her blessings.**

We are very much grateful to our beloved **Vice Chancellor Prof. Dr.RAJAT GUPTA,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. S P. CHOKKALINGAM, M.Tech., Ph.D., & Associate Dean, Dr. V. DHILIP KUMAR,M.E.,Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Professor & Head, Department of Artificial Intellegence & Machine Learning , Dr. S. ALEX DAVID, M.E., Ph.D.,** for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Internal Supervisor Dr.P.J.BESLIN PAJILA,M.E.,Ph.D .,** for his/her cordial support, valuable information and guidance, he/she helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Dr B Prabhu Shankar Ph.D** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

R.K.GOWTHAM	**(22UEAM0018)**
G.DHEEPAK	**(22UEAM0016)**
M.VADHANA KUMAR	**(22UEAM0063)**

ABSTRACT

Weather plays a crucial role in agricultural productivity, yet traditional forecasts often do not meet the specific needs of farmers. This project aims to bridge that gap by developing an agro-weather information system that provides real-time, crop-specific weather updates. By collecting and analyzing data from multiple sources, we generate tailored advisories that empower farmers to make informed decisions, enhance their agricultural practices, and mitigate weather-related risks. The user-friendly platform, accessible via SMS and mobile app, ensures timely dissemination of alerts about changing weather conditions and potential pest outbreaks. As a result, farmers receive accurate updates that lead to improved decision-making, increased crop yields, and reduced losses from adverse weather. Furthermore, the initiative emphasizes educational outreach through training sessions, enabling farmers to effectively interpret and utilize the weather data provided. Collaborating with agricultural institutions and meteorological agencies will help maximize the project's impact, contributing to sustainable and resilient agricultural practices in the long term.

Key Words: Precision Agriculture, Weather Forecasting, Crop Yield Prediction, Machine Learning Models, Satellite Data, Soil Moisture Monitoring, Climate Change Impact, Rainfall Prediction, Agro-Meteorology, Remote Sensing Technology, IoT in Agriculture, Weather Stations Seasonal Climate Patterns

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
AWIS	Agro Weather Information System
CRUD	Create, Read, Update, Delete
DBMS	Database Management System
ICT	Information and Communication Technology
IoT	Internet of Things
JSoN	JavaScript object Notation
ML	Machine Learning
NoSQL	Not only SQL
ORM	object-Relational Mapping
REST	Representational State Transfer
RTS	Real-Time Systems
SAAS	Software as a Service
SMS	Short Message Service
SQL	Structured Query Language
SSL	Secure Sockets Layer

TABLE OF CONTENTS

Page.No

Chapter 1

INTRODUCTION

1.1 Introduction

Agriculture is profoundly influenced by weather conditions, making accurate and timely weather information essential for farmers aiming to optimize crop yields and enhance their farming practices. Traditional weather forecasts, while useful, often lack the specificity required for agriculture, leaving farmers without the tailored guidance they need to navigate the complexities of weather variability. This gap in information can lead to poor decision-making, resulting in diminished crop production and financial losses.

The Agro-weather Information Project seeks to address this critical issue by providing real-time weather updates, crop-specific advisories, pest forecasts, and tailored recommendations based on current and predicted weather patterns. By integrating these services into mobile applications and SMS alerts, the project ensures that farmers receive crucial weather information directly on their devices, enabling them to act swiftly and effectively.

A key feature of the project is its user-friendly interface, designed to allow farmers to easily access not only real-time updates but also historical weather data. This historical context is invaluable for making informed decisions regarding planting, irrigation, and harvesting, as it allows farmers to analyze past weather trends and apply that knowledge to current conditions.

1.2 Aim of the project

The Agro-weather Information Project focuses on delivering precise and timely weather data tailored specifically for agricultural needs. By developing user-friendly tools, such as mobile apps and SMS alerts, the project ensures that farmers can easily access and utilize this vital information to enhance their decision-making processes. Accurate weather forecasts support informed farming practices, helping farmers plan activities like planting and irrigation effectively.

The system also aims to minimize agricultural risks by providing early warnings and alerts for extreme weather events, enabling farmers to take proactive measures to protect their crops. Furthermore, the project promotes sustainable agricultural practices by offering weather-based recommendations that optimize resource use and reduce environmental impact.

In addition to immediate benefits for farmers, the initiative seeks to enhance research and innovation in agriculture by providing comprehensive weather data, fostering a deeper understanding of climate patterns and their effects on farming. This holistic approach not only supports current agricultural practices but also contributes to the long-term sustainability and resilience of the agricultural sector.

1.3 Project Domain

The Agro-weather Information Project focuses on enhancing agricultural productivity by providing farmers with accurate, real-time weather data tailored to their specific needs. Recognizing that weather significantly impacts farming, the project aims to bridge the gap between traditional forecasts and the detailed information necessary for effective agricultural decision-making.

By utilizing data from various sources, including meteorological stations, satellites, and online weather services, the project offers insights into key weather parameters such as temperature, rainfall, humidity, wind speed, solar radiation, and soil moisture. This comprehensive information is delivered through user-friendly digital tools, including SMS alerts and mobile applications, ensuring farmers have timely access to critical updates.

The project also emphasizes education, conducting workshops and training sessions to empower farmers to interpret and apply weather data effectively. Collaboration with agricultural institutions, local governments, and meteorological agencies further enhances the initiative's reach and impact. Ultimately, the project aims to promote sustainable farming practices, reduce risks associated with climate variability, and support the livelihoods of farmers in regions where agriculture is a key economic activity.

1.4 Scope of the Project

The Agro-weather Information Project targets specific regions where agriculture plays a vital economic role, aiming for scalability based on available resources. By leveraging data from meteorological stations, satellites, and online weather services, the project provides comprehensive weather information covering essential parameters like temperature, rainfall, humidity, wind speed, solar radiation, and soil moisture.

To ensure farmers receive timely updates, the initiative will develop digital tools, including SMS services, for efficient dissemination of weather data. Educational outreach through training sessions and workshops will equip farmers with the skills to interpret and utilize this information effectively, enhancing their decision-making capabilities.

Collaboration with agricultural institutions, local governments, and meteorological agencies is crucial for maximizing the project's reach and impact. By fostering these partnerships, the project aims to create a robust support system that empowers farmers, improves agricultural productivity, and promotes sustainable practices in the targeted regions.

Chapter 2

LITERATURE REVIEW

2.1 Literature Review

1. Author Name: C. Zoremsanga and J. Hussain Paper Title: ”Particle Swarm optimized Deep Learning Models for Rainfall Prediction: A Case Study in Aizawl, Mizoram” Publication Details: IEEE Access, vol. 12, pp. 57172-57184 Year of Publication: 2024 Main Content of Paper: This paper discusses the use of Particle Swarm optimization (PSo) integrated with deep learning models to predict rainfall in Aizawl, Mizoram. The study demonstrates the effectiveness of PSo in optimizing the deep learning models for improved accuracy in rainfall prediction.

2. Author Name: E. Rocha Rodrigues, I. oliveira, R. Cunha, and M. Netto Paper Title: ”DeepDownscale: A Deep Learning Strategy for High-Resolution Weather Forecast” Publication Details: 2018 IEEE 14th International Conference on e-Science (e-Science), Amsterdam, Netherlands, pp. 415-422 Year of Publication: 2018 Main Content of Paper: This paper introduces DeepDownscale, a deep learning strategy aimed at providing high-resolution weather forecasts. The approach focuses on enhancing the precision of weather predictions by downscaling coarse-grained meteorological data.

3. Author Name: M. M. Hassan et al. Paper Title: ”Machine Learning-Based Rainfall Prediction: Unveiling Insights and Forecasting for Improved Preparedness” Publication Details: IEEE Access, vol. 11, pp. 132196-132222 Year of Publication: 2023 Main Content of Paper: This paper explores various machine learning techniques for predicting rainfall, providing insights into their effectiveness and forecasting capabilities. The study emphasizes the importance of accurate predictions for improving preparedness in the face of weather-related challenges.

4. Author Name: R. Cai, S. Xie, B. Wang, R. Yang, D. Xu, and Y. He Paper Title: ”Wind Speed Forecasting Based on Extreme Gradient Boosting” Publication Details: IEEE Access, vol. 8, pp. 175063-175069 Year of Publication: 2020 Main Content of Paper: This paper presents a wind speed forecasting model based on Extreme Gradient Boosting (XGBoost). The research highlights the effectiveness

of XGBoost in predicting wind speeds with high accuracy, contributing to better management of wind energy resources.

2.2 Gap Identification

1.C. Zoremsanga and J. Hussain (2024): While the paper demonstrates the effectiveness of Particle Swarm optimization (PSo) in optimizing deep learning models for rainfall prediction, it may not address the generalizability of the PSo-optimized models across different geographic regions or climates. Future research could focus on adapting the methodology for various locales with differing meteorological conditions.

2.E. Rocha Rodrigues et al. (2018): The DeepDownscale approach enhances precision but may not sufficiently investigate the impact of different data sources or the influence of seasonal variations on model performance. Future studies could examine the robustness of the deep learning strategy under varying data conditions or explore ensemble methods that combine predictions from multiple models.

3.M. M. Hassan et al. (2023): This paper provides insights into various machine learning techniques for rainfall prediction but may lack a comprehensive comparative analysis of their performance in real-time forecasting scenarios. Further research could involve developing hybrid models that integrate multiple machine learning approaches for improved predictive accuracy and reliability in dynamic weather situations.

4.R. Cai et al. (2020): While the study shows the effectiveness of XGBoost for wind speed forecasting, it may not address the potential impact of extreme weather events or rapid changes in atmospheric conditions on the accuracy of predictions. Future research could explore adaptive models that account for sudden shifts in weather patterns and their effects on wind speed predictions.

General Research Gaps Across the Studies:

- Integration of Models: There is a lack of studies that explore the integration of different machine learning and optimization techniques (like PSo, XGBoost) for multi-faceted weather prediction tasks.
- Real-Time Application: Most studies focus on model development and accuracy but do not sufficiently address the implementation of these models in real-time forecasting and decision-making frameworks.

- Long-Term Predictions: There's a need for research focusing on the sustainability and accuracy of predictions over longer periods, particularly with climate change impacts.
- Data Quality and Sources: Many studies rely on specific datasets, and future research could investigate the impact of data quality, availability, and granularity on model performance across different weather prediction scenarios.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The current system used by farmers for accessing weather information primarily relies on generic weather forecasts provided by mainstream services like radio, television, or mobile applications. These services offer broad regional forecasts that do not account for localized weather variations specific to the farming areas. As a result, farmers often make decisions based on outdated or irrelevant weather data, which may not be suitable for their specific crops or farming schedules.

The major disadvantages of the existing system include the lack of real-time updates, minimal accessibility for rural farmers, and the absence of tailored recommendations for different crops. Farmers cannot receive accurate, localized forecasts for planning critical activities such as irrigation, fertilization, or pest management. This leads to decreased agricultural productivity and inefficient use of resources.

3.2 Problem Statement

The Agro-Weather Information System (AWIS) is designed to overcome the challenges faced by rural farmers by providing precise, localized weather information. The proposed system will integrate advanced machine learning models, such as XGBoost and LightGBM, to analyze both historical and real-time weather data, offering farmers crop-specific advice, pest management alerts, and optimized farming schedules. The system will be available via mobile applications and SMS services, ensuring ease of use for rural farmers.

The advantages of the proposed system include accurate, real-time weather forecasts, automated crop-specific recommendations, and a user-friendly interface. The integration with meteorological agency APIs ensures a continuous flow of weather data, while the machine learning models provide reliable, predictive insights. This system aims to enhance agricultural productivity and resource management.

3.3 System Specification

3.3.1 Hardware Specification

- Processor: Intel Core i7 10th Gen or AMD Ryzen 7 (or newer)
- RAM: 16 GB DDR4
- Storage: 512 GB SSD
- Graphics: NVIDIA GTX 1650 or equivalent
- Network: WiFi 6, 802.11ax Wireless LAN

3.3.2 Software Specification

- operating System: Windows 11, Ubuntu 20.04, macoS 12 (or newer)
- Programming Languages: Python 3.9+, JavaScript (Node.js)
- Database: MongoDB (NoSQL), MySQL
- Machine Learning Frameworks: Scikit-Learn, XGBoost, LightGBM
- IDE: Visual Studio Code, Jupyter Notebook, Anaconda

3.3.3 Standards and Policies

Anaconda Prompt

Anaconda Prompt is a command-line interface that deals explicitly with Machine Learning modules. It is available on Windows, Linux, and macoS. Anaconda Prompt provides access to multiple IDEs that make coding more manageable, and it allows Python UI implementations.

Standard Used: ISo/IEC 27001

Jupyter Notebook

Jupyter is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It is widely used for data cleaning, numerical simulation, statistical modeling, data visualization, and machine learning.

Standard Used: ISo/IEC 27001

MongoDB

MongoDB is a NoSQL database used for storing and managing real-time and historical weather data in the AWIS. It provides high flexibility and scalability, making it ideal for handling large datasets.

Standard Used: ISo/IEC 27017

LightGBM

LightGBM is a machine learning framework optimized for efficiency in handling large datasets. It is used in AWIS for predictive modeling, offering faster training times and higher accuracy.

Standard Used: ISo/IEC 25010

Chapter 4

METHODOLOGY

4.1 Proposed System

The Agro-Weather Information System is designed to address the challenges faced by farmers, particularly in rural areas, by providing them with precise and localized weather forecasts, crop advisories, pest predictions, and tailored agricultural recommendations. The ultimate goal of this system is to enhance agricultural productivity, promote sustainability, and reduce the risks associated with unpredictable weather patterns. By empowering farmers with timely and relevant information, the system aims to improve decision-making in agricultural practices, minimize crop loss, and increase overall farm efficiency.

The core of the system revolves around the integration of multiple real-time data sources, including meteorological data, satellite imagery, and ground-based sensors. These data sources work together to provide highly localized weather forecasts, which are essential for farmers who rely on accurate weather information for critical decisions, such as when to plant, irrigate, fertilize, and harvest crops. Traditional weather forecasts are often too general and not precise enough for small-scale farmers who need hyper-local data. By offering village-specific weather updates, the system helps farmers prepare for adverse weather conditions, such as droughts, floods, or unseasonal rains, which can significantly impact crop yield.

In addition to weather forecasts, the system provides crop advisories that are tailored to the specific needs of the region and the type of crops being cultivated. This feature leverages the expertise of agricultural scientists and local agronomists, who collaborate to generate recommendations based on current weather conditions, soil moisture levels, and pest infestation patterns. These crop advisories provide practical guidance on fertilizer application, pest control measures, irrigation schedules, and best practices for improving soil health. By offering real-time, actionable advice, the system helps farmers optimize their crop management strategies and increase their chances of a successful harvest.

A key component of the system is its pest prediction model, which analyzes environmental factors such as humidity, temperature, and rainfall to forecast pest outbreaks. This early warning system allows farmers to take preventive measures to protect their crops from pest attacks before they become widespread. By receiving timely pest alerts, farmers can reduce their reliance on chemical pesticides, which not only saves costs but also promotes environmentally friendly farming practices. This aspect of the system contributes to the sustainability of agricultural operations by minimizing the negative impact of chemical inputs on the ecosystem.

The Agro-Weather Information System is designed to be user-friendly, with a simple interface accessible through mobile devices. Given the growing penetration of smartphones in rural areas, the system can reach a wide audience of farmers. The user interface is available in multiple local languages to ensure inclusivity and ease of use. Farmers can receive daily weather updates, advisories, and pest alerts via SMS, mobile applications, or voice-based systems, ensuring that even those with limited digital literacy can benefit from the system.

Collaboration with local agricultural experts and institutions is a critical aspect of the system. By involving agronomists, meteorologists, and extension workers, the system ensures that the information provided to farmers is accurate, context-specific, and grounded in expert knowledge. Regular updates and feedback loops allow the system to continuously evolve and adapt to the changing needs of the farming community.

In conclusion, the Agro-Weather Information System offers a comprehensive solution to the challenges faced by rural farmers in managing their crops amid uncertain weather patterns and pest threats. By combining real-time data, expert insights, and user-friendly technology, the system empowers farmers to make informed decisions, reduce risks, and ultimately improve agricultural productivity and sustainability.

4.2 General Architecture

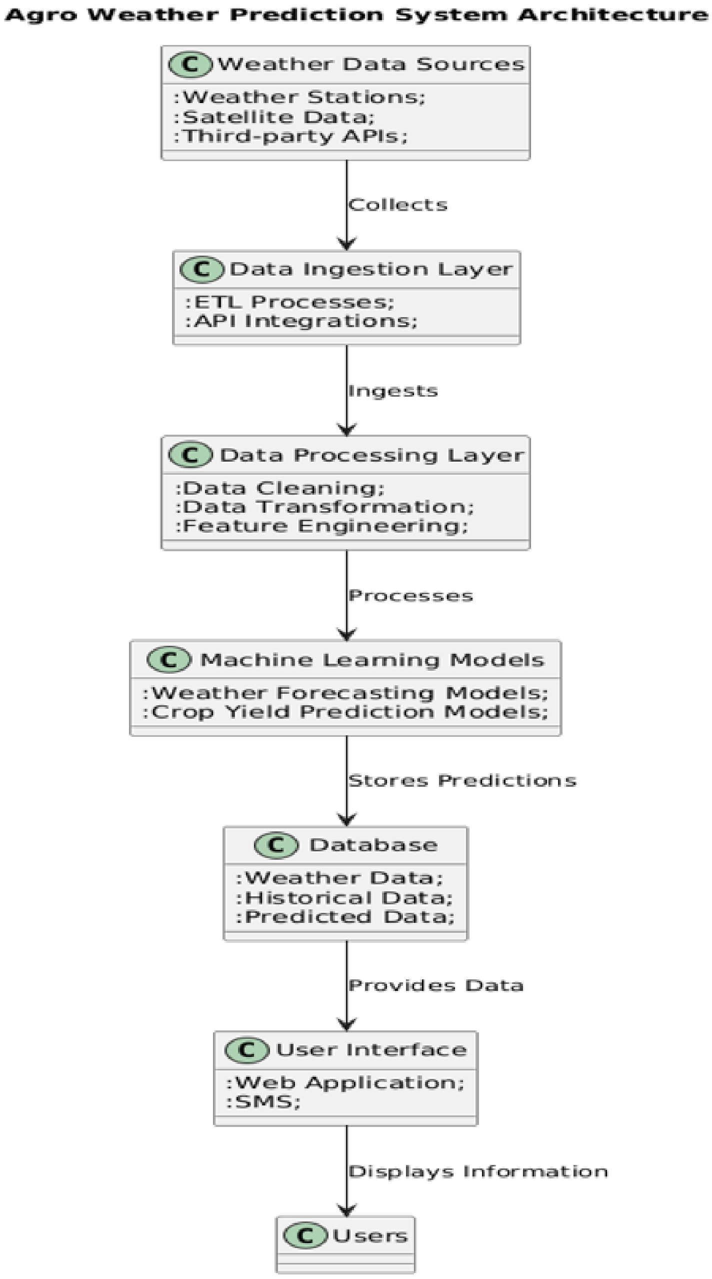


Figure 4.1: **Architecture Diagram**

4.2.1 Description

The architecture diagram outlines the overall structure of the Agro-Weather Information System, highlighting the key components and their interactions. It typically includes:

User Interface: This is the front-end component where farmers can interact with the system. It provides functionalities like requesting weather updates, crop-specific advisories, pest forecasts, and tailored recommendations.

Agro-Weather System: The core component responsible for processing user requests and managing the workflow between different modules.

Weather Data Source: An external API that fetches real-time weather data based on the user's city or location. It provides essential weather parameters needed for analysis and advisories.

Weather Database: A repository that stores historical weather data, crop advisories, pest forecasts, and recommendations. It supports the system in providing accurate insights and forecasts.

Collaborative Modules: These may include integrations with meteorological agencies, agronomists, and SMS services to enhance the system's capabilities in delivering timely and relevant information to the users.

4.3 Design Phase

4.3.1 Data Flow Diagram

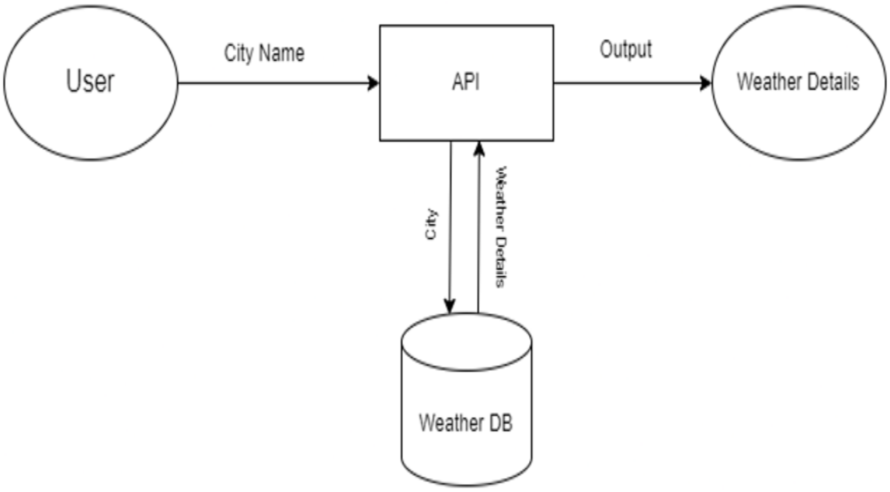


Figure 4.2: **Data Flow Diagram**

Description:

The data flow diagram illustrates how data moves within the Agro-Weather Information System, detailing the flow between various components. The main elements in the diagram include:

User: Initiates requests to the system, such as seeking weather updates or forecasts.

City Name Input: The user inputs their city name, which serves as the basis for fetching localized weather information.

API: Represents the connection to an external weather data source that processes the city name and retrieves the relevant weather details. This API serves as the intermediary between the user requests and the weather data retrieval.

Weather Details: The output from the API, which includes essential weather information (e.g., temperature, humidity, rainfall) relevant to the user's location.

Weather Database: An optional data store that may hold both current and historical weather information. It supports the system in providing comprehensive insights and allows for future requests to be serviced quickly without having to call the API every time.

overall, the data flow diagram effectively maps the interactions between users and the system components, providing a clear view of how information is processed and exchanged to fulfill user needs.

4.3.2 Use Case Diagram

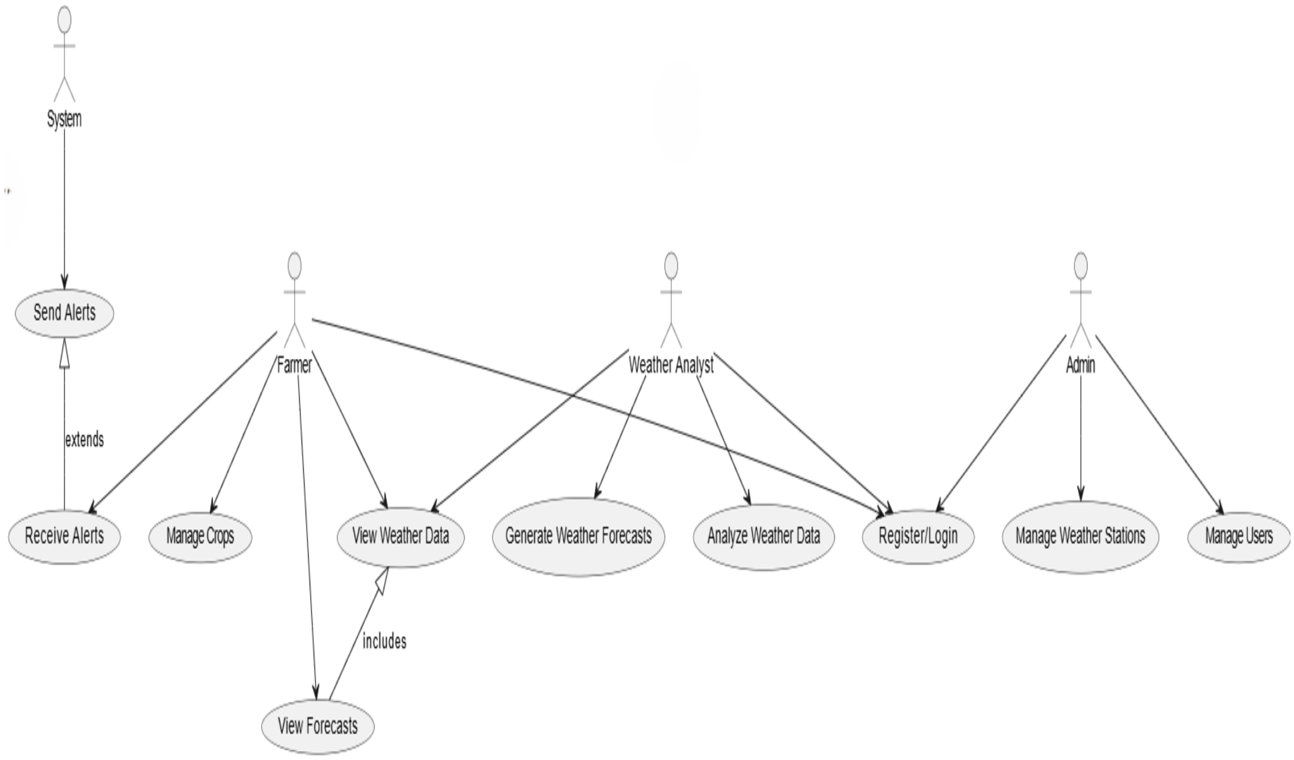


Figure 4.3: **Use Case Diagram**

Description:

The use case diagram for the Agro-Weather Information System illustrates the various interactions between the users (primarily farmers) and the system's functionalities. It highlights the key use cases and the actors involved. The main components include:

Actors:

Farmer: The primary user of the system who seeks weather information and agricultural advisories. Use Cases:

Request Weather Update: The farmer can request real-time weather updates based on their location. This use case involves fetching and displaying the latest weather conditions.

Request Crop-Specific Advisory: The farmer can seek tailored advice regarding specific crops. This use case allows the farmer to receive recommendations based on current weather and agricultural best practices.

Request Pest Forecast: The farmer can inquire about potential pest threats based on weather patterns and seasonal data. This use case aims to provide timely warnings and preventive measures.

Request Tailored Recommendations: The farmer can receive customized suggestions based on their location, current weather, and crop types. This could include planting schedules, pest management strategies, and other relevant tips.

Access Historical Data: The farmer can view historical weather data, crop advisories, and pest forecasts. This use case allows farmers to analyze past trends for better decision-making in future farming activities.

System: Represented as a boundary that encapsulates all the use cases and interactions, indicating that the system handles the execution of these requests and returns relevant information to the farmer.

4.3.3 Class Diagram

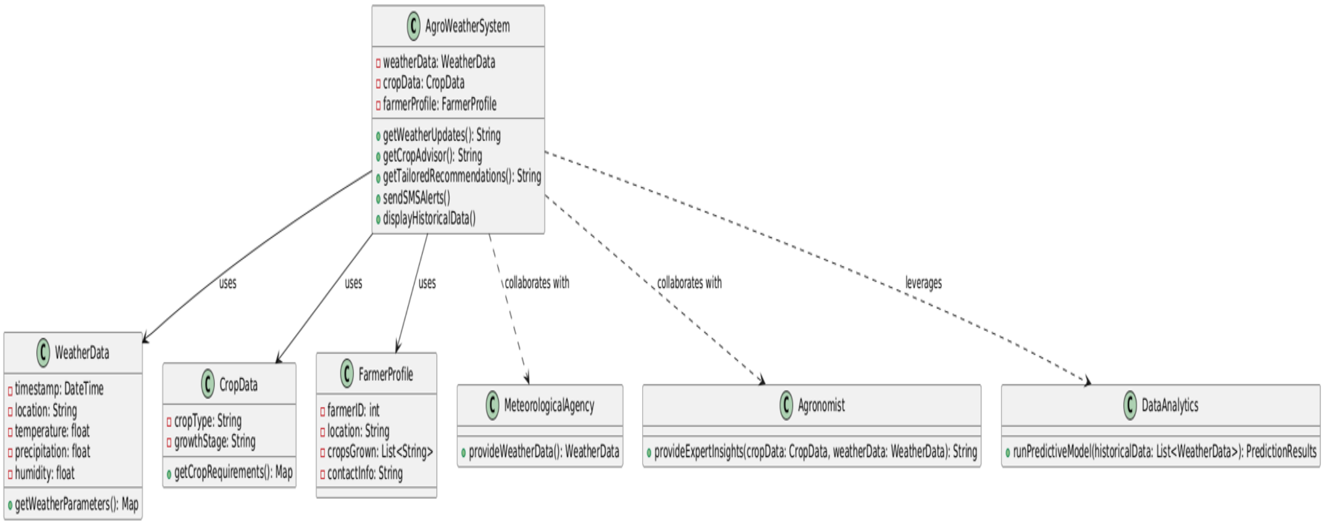


Figure 4.4: **Class Diagram**

Description:
The class diagram illustrates the structure of classes within the Agro-Weather System, defining their attributes and methods.

Classes:

Farmer: Attributes include FarmerID, Name, Phone, and Email, with methods for requesting weather updates, crop advisories, tailored recommendations, and accessing historical data. WeatherData: Attributes such as WeatherID, Date, Temperature, Humidity, Rainfall, and WindSpeed, with a method to fetch data. CropAdvisory: Contains AdvisoryID, CropType, Advice, and Date, with a method to fetch advisory data. Recommendation: Includes RecommendationID, RecommendationType, Content, and Date, with a method to generate recommendations. HistoricalData: Attributes like DataID, Date, and DataContent, with a method to fetch historical data. MeteorologicalAgency: Attributes include AgencyID, Name, and ContactInfo, with a method to provide weather data. Agronomist: Contains AgronomistID, Name, Expertise, and ContactInfo, with a method to provide recommendations. SMSService: Includes a method for sending SMS alerts. UserInterface: Contains methods for displaying various data types (weather updates, crop advisories, recommendations, and historical data). Relationships:

Farmers receive multiple instances of WeatherData, CropAdvisory, Recommendation, and access HistoricalData. WeatherData is provided by a Meteorological Agency, while Crop Advisory and Pest Forecast are created by Agronomists. Various data types are displayed on the User Interface and sent via the SMS service.

4.3.4 Sequence Diagram

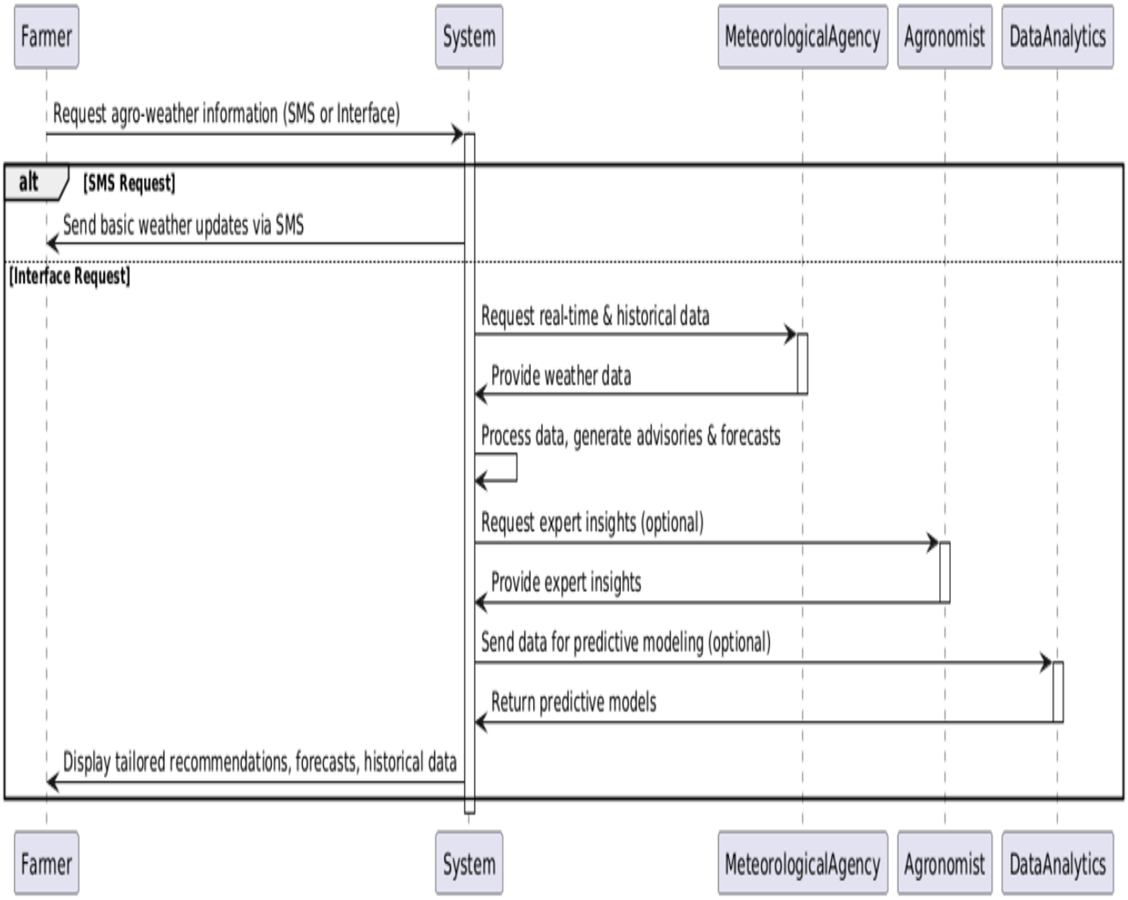

Farmer
System
MeteorologicalAgency
Agronomist
DataAnalytics
Request agro-weather information (SMS or Interface)
alt
[SMS Request]
Send basic weather updates via SMS
[Interface Request]
Request real-time & historical data
Provide weather data
Process data, generate advisories & forecasts
Request expert insights (optional)
Provide expert insights
Send data for predictive modeling (optional)
Return predictive models
Display tailored recommendations, forecasts, historical data
Farmer
System
MeteorologicalAgency
Agronomist
DataAnalytics

Figure 4.5: **Sequence Diagram**

Description:

The sequence diagram illustrates the interactions between a farmer and the Agro-Weather System, highlighting the communication flow for various requests.

Request Weather Update: The farmer initiates a request for a weather update, prompting the system to fetch real-time weather data from the Weather Data Source. Upon retrieval, the system sends notifications through the Mobile App and SMS service.

Request Crop-Specific Advisory: When the farmer seeks crop-specific advice, the system contacts the Crop Advisory Service, receives the advisory, and communicates this to the farmer via mobile and SMS.

Request Pest Forecast: The farmer requests pest forecast information. The system fetches data from the Pest Forecast System and notifies the farmer through both mobile and SMS.

Request Tailored Recommendations: For tailored recommendations, the system collaborates with the Meteorological Agency for data, requests insights from an Agronomist, processes the recommendations, and delivers them via mobile and SMS.

This diagram effectively captures the essential functions and response mechanisms of the Agro-Weather System as it serves the farmer's needs.

4.3.5 Collaboration diagram

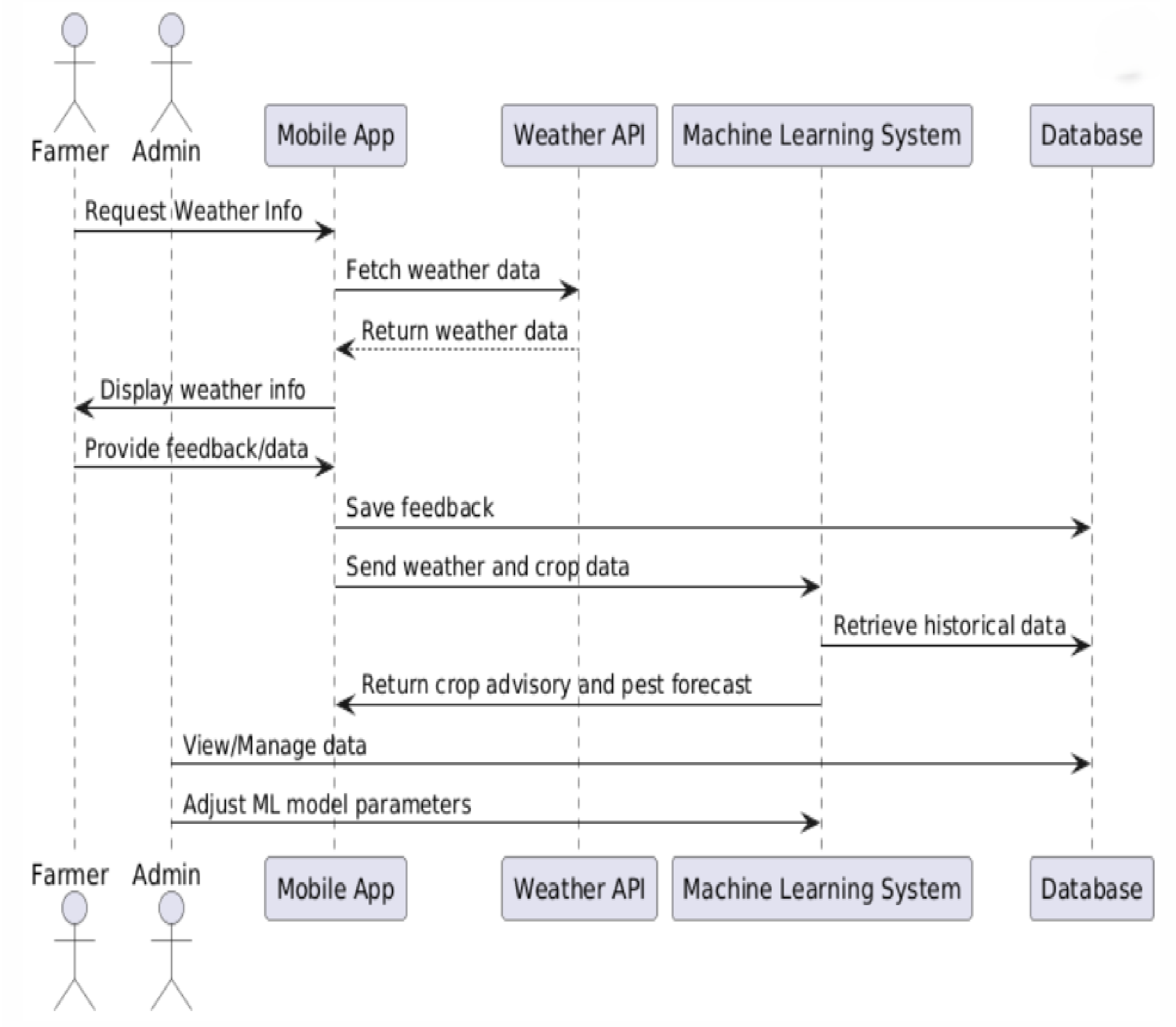


Figure 4.6: **collabration diagram**

Description:

The collaboration diagram illustrates the interactions between various actors and components within the Agro-Weather Information System. It showcases the flow of information and the roles played by different entities in delivering weather updates, crop advisories, and pest forecasts to farmers. The key components and interactions are as follows:

Actors: Farmer: The primary user of the system who seeks weather information and provides feedback. Admin: An administrative user responsible for managing data and adjusting machine learning model parameters. Participants: Mobile App: The user interface that enables farmers to interact with the system and access relevant information. Weather API: An external service that provides real-time weather data based on user requests. Machine Learning System (ML System): A component that analyzes data to provide crop advisories and pest forecasts. Database (DB): A central repository for storing weather data, user feedback, historical data, and machine learning model parameters. Interactions: Weather Information Request:

The farmer initiates a request for weather information through the Mobile App. The Mobile App sends a request to the Weather API to fetch the relevant weather data. Upon retrieving the data, the Weather API responds with the weather details, which are then displayed to the farmer through the Mobile App. Feedback Submission:

The farmer can provide feedback or additional data through the Mobile App. This feedback is saved in the Database for future reference and analysis. Machine Learning Integration:

The Mobile App sends weather and crop data to the ML System for analysis. The ML System retrieves historical data from the Database to enhance the accuracy of its recommendations. The ML System returns crop advisories and pest forecasts back to the Mobile App, which are then presented to the farmer. Administrative Management:

The Admin can access the Database to view and manage data, ensuring the system is up-to-date and accurate. The Admin also has the capability to adjust parameters of the ML System to optimize its performance and improve advisory outputs.

4.3.6 Activity Diagram

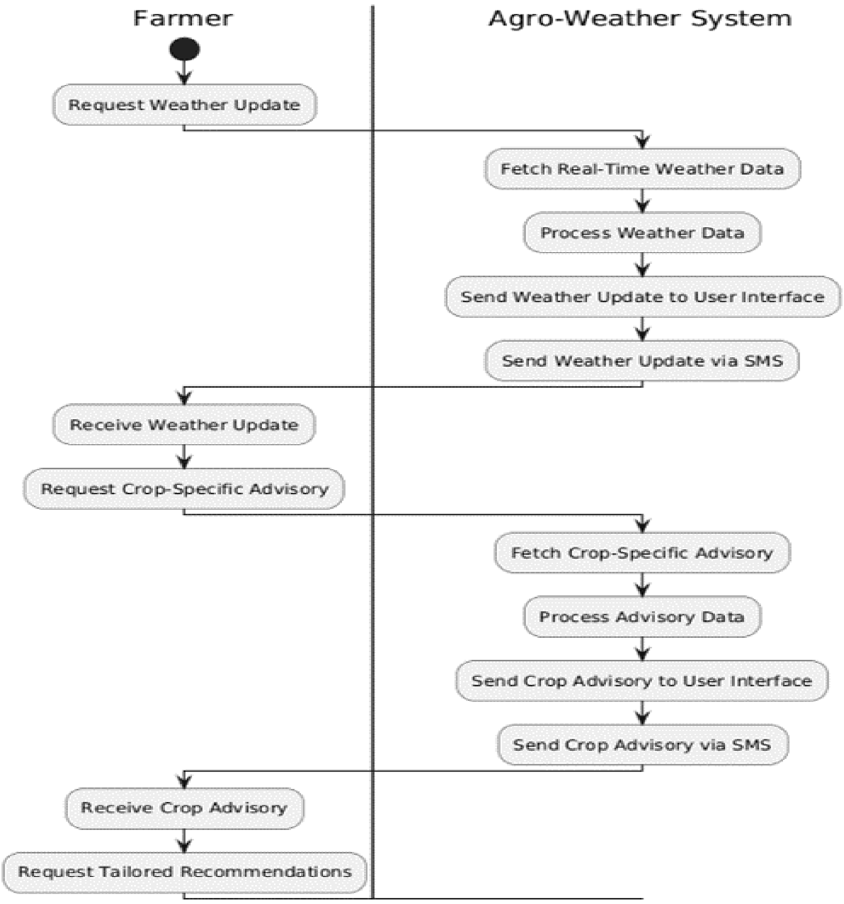

Farmer
Agro-Weather System
Request Weather Update
Fetch Real-Time Weather Data
Process Weather Data
Send Weather Update to User Interface
Send Weather Update via SMS
Receive Weather Update
Request Crop-Specific Advisory
Fetch Crop-Specific Advisory
Process Advisory Data
Send Crop Advisory to User Interface
Send Crop Advisory via SMS
Receive Crop Advisory
Request Tailored Recommendations

Figure 4.7: **Activity Diagram**

Description:

The diagram illustrates the interaction between a farmer and an Agro-Weather Information System in a structured process, showcasing the flow of information for obtaining weather updates, crop-specific advisories, and tailored recommendations. It is divided into two main entities: the Farmer on the left and the Agro-Weather System on the right.

1.Request Weather Update : The process begins when the farmer requests a weather update. The system fetches real-time weather data, processes it, and sends the update back to the farmer via the user interface and SMS.

2.Receive Weather Update : After receiving the weather update, the farmer may request crop-specific advisories based on the current conditions.

3.Request Crop-Specific Advisory : The system responds by fetching relevant crop advisory data, processing it, and delivering it to the farmer through the same channels.

4.Receive Crop Advisory: Upon receiving the advisory, the farmer can then request more tailored recommendations, which are customized according to the specific agricultural needs of the farmer's crops and weather conditions.

This diagram efficiently demonstrates the sequential interaction between the farmer and the system, ensuring that vital information like weather forecasts and crop advisories is communicated in a clear, timely, and actionable manner. The system relies on real-time data and a user-friendly interface to keep farmers informed, helping them make informed decisions.

4.4 Algorithm & Pseudo Code

4.4.1 Algorithm

The K-means clustering algorithm is utilized to classify data points into clusters. This algorithm minimizes the distance between points in the same cluster by iterating over the dataset and recalculating centroids. The steps for the algorithm are as follows:

1. Select k initial centroids randomly from the dataset.
2. For each data point, calculate the Euclidean distance from each centroid.
3. Assign each data point to the nearest centroid.
4. Update the centroids by calculating the mean of all data points assigned to each centroid.
5. Repeat the process until the centroids no longer change or the specified number of iterations is reached.

4.4.2 Pseudo Code

The pseudocode for the K-means algorithm is outlined below:

K-means Algorithm Dataset $S = \{x_1, x_2, \ldots, x_h\}$ Vector u indicating the cluster label for each sample Input: Dataset $S = \{x_1, x_2, \ldots, x_h\}$

Procedure:

```
Select $k$ samples from dataset $S$ as the initial centroid vector $c = \{c_1, c_2, \ldots, c_k\}$
For each sample $x_i$ then
Calculate the Euclidean distance between $x_i$ and $c_j$ (for $j = 1, 2, \ldots, k$)
```

```
4: Identify the nearest centroid $c_j$ and assign $u_i = j$, where $u_i$ represents the cluster corresponding to $x_i$
5: End for
6: $u = \{u_1, u_2, \ldots, u_h\}$
7: For each cluster then
8: Compute the centroid of each cluster $v_i$
9: End for
10: $v = \{v_1, v_2, \ldots, v_k\}, u = \{u_1, u_2, \ldots, u_h\}$
11: If $v == c$ then
12: Return $u$
13: Else
14: $c = v$ and return to step 2
15: End if
output: Vector $u$ indicating the cluster label for each sample
```

4.4.3 Data Set

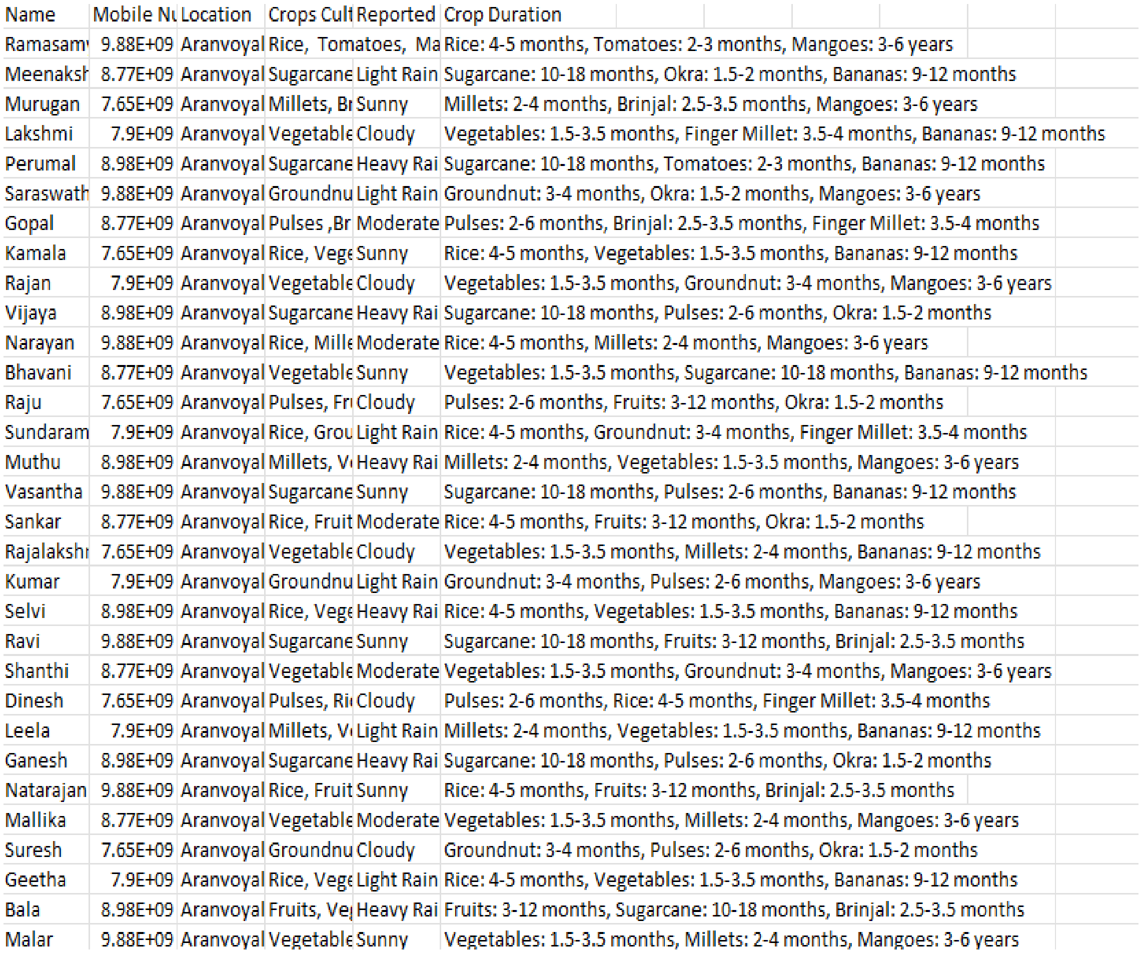
The dataset for this project consists of historical and real-time meteorological data, including temperature, humidity, rainfall, and wind speed. The data is collected from a meteorological agency API integrated into the Agro-Weather Information System (AWIS). Additionally, the dataset includes agricultural data such as crop type, soil moisture, and pest outbreaks. This combination of weather and agricultural data allows the system to provide precise crop advisories and pest forecasts tailored to specific farming conditions.

4.5 Module Description

4.5.1 Module1: Data Collection and Integration

This module integrates real-time weather data from meteorological agency APIs and historical datasets related to crop production. The data is preprocessed and cleaned to ensure consistency before being passed to the predictive models. It ensures that both historical and current weather data are accessible for generating real-time advisories.

4.5.2 Data Collection



Name	Mobile Nu	Location	Crops Cult	Reported	Crop Duration
Ramasam	9.88E+09	Aranvoyal	Rice, Tomatoes, Ma		Rice: 4-5 months, Tomatoes: 2-3 months, Mangoes: 3-6 years
Meenaksh	8.77E+09	Aranvoyal	Sugarcane	Light Rain	Sugarcane: 10-18 months, Okra: 1.5-2 months, Bananas: 9-12 months
Murugan	7.65E+09	Aranvoyal	Millets, Br	Sunny	Millets: 2-4 months, Brinjal: 2.5-3.5 months, Mangoes: 3-6 years
Lakshmi	7.9E+09	Aranvoyal	Vegetable	Cloudy	Vegetables: 1.5-3.5 months, Finger Millet: 3.5-4 months, Bananas: 9-12 months
Perumal	8.98E+09	Aranvoyal	Sugarcane	Heavy Rai	Sugarcane: 10-18 months, Tomatoes: 2-3 months, Bananas: 9-12 months
Saraswath	9.88E+09	Aranvoyal	Groundnu	Light Rain	Groundnut: 3-4 months, Okra: 1.5-2 months, Mangoes: 3-6 years
Gopal	8.77E+09	Aranvoyal	Pulses ,Br	Moderate	Pulses: 2-6 months, Brinjal: 2.5-3.5 months, Finger Millet: 3.5-4 months
Kamala	7.65E+09	Aranvoyal	Rice, Vege	Sunny	Rice: 4-5 months, Vegetables: 1.5-3.5 months, Bananas: 9-12 months
Rajan	7.9E+09	Aranvoyal	Vegetable	Cloudy	Vegetables: 1.5-3.5 months, Groundnut: 3-4 months, Mangoes: 3-6 years
Vijaya	8.98E+09	Aranvoyal	Sugarcane	Heavy Rai	Sugarcane: 10-18 months, Pulses: 2-6 months, Okra: 1.5-2 months
Narayan	9.88E+09	Aranvoyal	Rice, Mille	Moderate	Rice: 4-5 months, Millets: 2-4 months, Mangoes: 3-6 years
Bhavani	8.77E+09	Aranvoyal	Vegetable	Sunny	Vegetables: 1.5-3.5 months, Sugarcane: 10-18 months, Bananas: 9-12 months
Raju	7.65E+09	Aranvoyal	Pulses, Fr	Cloudy	Pulses: 2-6 months, Fruits: 3-12 months, Okra: 1.5-2 months
Sundaram	7.9E+09	Aranvoyal	Rice, Grou	Light Rain	Rice: 4-5 months, Groundnut: 3-4 months, Finger Millet: 3.5-4 months
Muthu	8.98E+09	Aranvoyal	Millets, V	Heavy Rai	Millets: 2-4 months, Vegetables: 1.5-3.5 months, Mangoes: 3-6 years
Vasantha	9.88E+09	Aranvoyal	Sugarcane	Sunny	Sugarcane: 10-18 months, Pulses: 2-6 months, Bananas: 9-12 months
Sankar	8.77E+09	Aranvoyal	Rice, Fruit	Moderate	Rice: 4-5 months, Fruits: 3-12 months, Okra: 1.5-2 months
Rajalaksh	7.65E+09	Aranvoyal	Vegetable	Cloudy	Vegetables: 1.5-3.5 months, Millets: 2-4 months, Bananas: 9-12 months
Kumar	7.9E+09	Aranvoyal	Groundnu	Light Rain	Groundnut: 3-4 months, Pulses: 2-6 months, Mangoes: 3-6 years
Selvi	8.98E+09	Aranvoyal	Rice, Vege	Heavy Rai	Rice: 4-5 months, Vegetables: 1.5-3.5 months, Bananas: 9-12 months
Ravi	9.88E+09	Aranvoyal	Sugarcane	Sunny	Sugarcane: 10-18 months, Fruits: 3-12 months, Brinjal: 2.5-3.5 months
Shanthi	8.77E+09	Aranvoyal	Vegetable	Moderate	Vegetables: 1.5-3.5 months, Groundnut: 3-4 months, Mangoes: 3-6 years
Dinesh	7.65E+09	Aranvoyal	Pulses, Ri	Cloudy	Pulses: 2-6 months, Rice: 4-5 months, Finger Millet: 3.5-4 months
Leela	7.9E+09	Aranvoyal	Millets, V	Light Rain	Millets: 2-4 months, Vegetables: 1.5-3.5 months, Bananas: 9-12 months
Ganesh	8.98E+09	Aranvoyal	Sugarcane	Heavy Rai	Sugarcane: 10-18 months, Pulses: 2-6 months, Okra: 1.5-2 months
Natarajan	9.88E+09	Aranvoyal	Rice, Fruit	Sunny	Rice: 4-5 months, Fruits: 3-12 months, Brinjal: 2.5-3.5 months
Mallika	8.77E+09	Aranvoyal	Vegetable	Moderate	Vegetables: 1.5-3.5 months, Millets: 2-4 months, Mangoes: 3-6 years
Suresh	7.65E+09	Aranvoyal	Groundnu	Cloudy	Groundnut: 3-4 months, Pulses: 2-6 months, Okra: 1.5-2 months
Geetha	7.9E+09	Aranvoyal	Rice, Vege	Light Rain	Rice: 4-5 months, Vegetables: 1.5-3.5 months, Bananas: 9-12 months
Bala	8.98E+09	Aranvoyal	Fruits, Ve	Heavy Rai	Fruits: 3-12 months, Sugarcane: 10-18 months, Brinjal: 2.5-3.5 months
Malar	9.88E+09	Aranvoyal	Vegetable	Sunny	Vegetables: 1.5-3.5 months, Millets: 2-4 months, Mangoes: 3-6 years

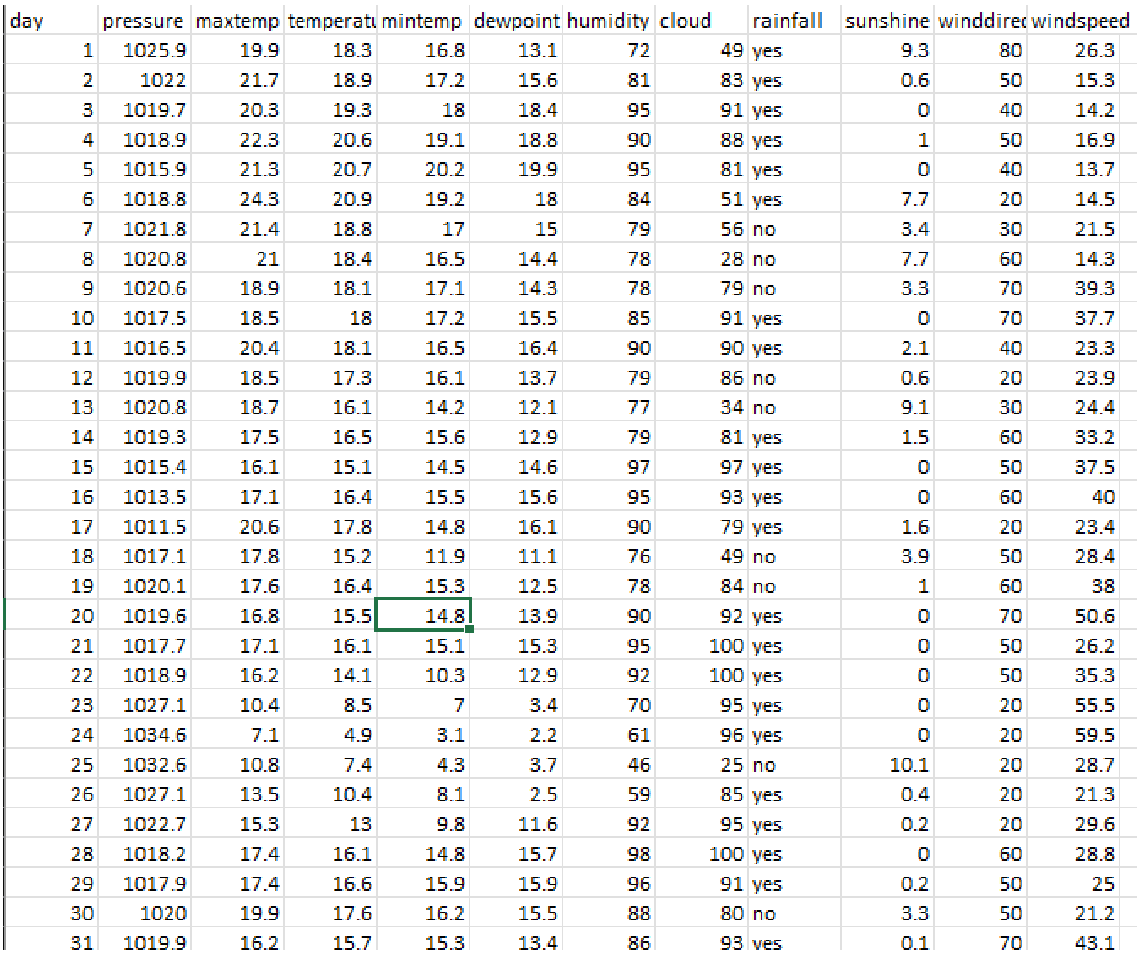
Figure 4.8: **Collection of Data**

In figure 4.8 the collection of datasets which contains the name,mobile number,location,crops cultivated,reported crops,crop duration to gather their information as an dataset to produce relevant data to the farmers

4.5.3 Module2: Machine Learning Model Implementation

This module is responsible for implementing the machine learning algorithms such as XGBoost and LightGBM. These algorithms are trained on the collected datasets to predict weather patterns, crop yield forecasts, and potential pest outbreaks. The module includes hyperparameter tuning for optimizing model performance.

4.5.4 Predicted Data

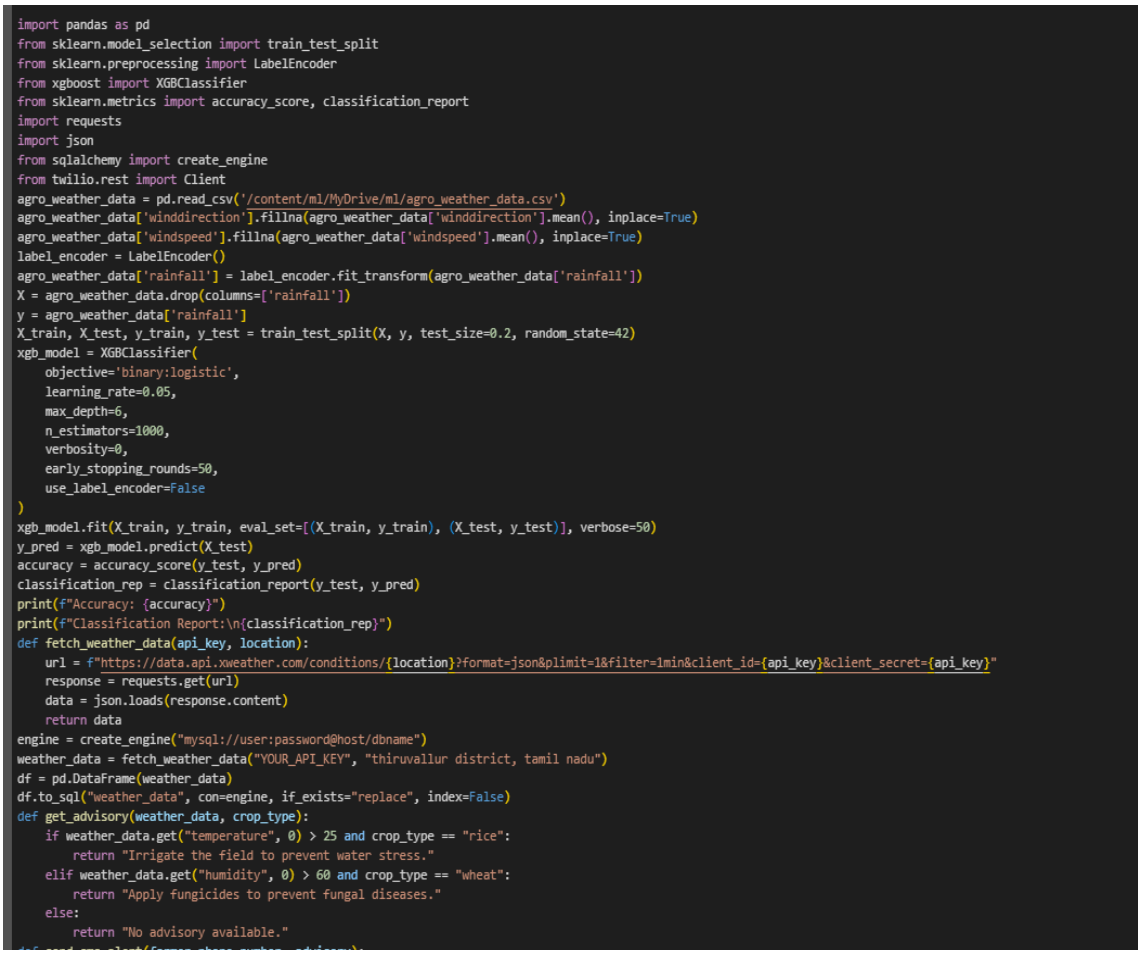


day	pressure	maxtemp	temperatu	mintemp	dewpoint	humidity	cloud	rainfall	sunshine	winddire	windspeed
1	1025.9	19.9	18.3	16.8	13.1	72	49	yes	9.3	80	26.3
2	1022	21.7	18.9	17.2	15.6	81	83	yes	0.6	50	15.3
3	1019.7	20.3	19.3	18	18.4	95	91	yes	0	40	14.2
4	1018.9	22.3	20.6	19.1	18.8	90	88	yes	1	50	16.9
5	1015.9	21.3	20.7	20.2	19.9	95	81	yes	0	40	13.7
6	1018.8	24.3	20.9	19.2	18	84	51	yes	7.7	20	14.5
7	1021.8	21.4	18.8	17	15	79	56	no	3.4	30	21.5
8	1020.8	21	18.4	16.5	14.4	78	28	no	7.7	60	14.3
9	1020.6	18.9	18.1	17.1	14.3	78	79	no	3.3	70	39.3
10	1017.5	18.5	18	17.2	15.5	85	91	yes	0	70	37.7
11	1016.5	20.4	18.1	16.5	16.4	90	90	yes	2.1	40	23.3
12	1019.9	18.5	17.3	16.1	13.7	79	86	no	0.6	20	23.9
13	1020.8	18.7	16.1	14.2	12.1	77	34	no	9.1	30	24.4
14	1019.3	17.5	16.5	15.6	12.9	79	81	yes	1.5	60	33.2
15	1015.4	16.1	15.1	14.5	14.6	97	97	yes	0	50	37.5
16	1013.5	17.1	16.4	15.5	15.6	95	93	yes	0	60	40
17	1011.5	20.6	17.8	14.8	16.1	90	79	yes	1.6	20	23.4
18	1017.1	17.8	15.2	11.9	11.1	76	49	no	3.9	50	28.4
19	1020.1	17.6	16.4	15.3	12.5	78	84	no	1	60	38
20	1019.6	16.8	15.5	14.8	13.9	90	92	yes	0	70	50.6
21	1017.7	17.1	16.1	15.1	15.3	95	100	yes	0	50	26.2
22	1018.9	16.2	14.1	10.3	12.9	92	100	yes	0	50	35.3
23	1027.1	10.4	8.5	7	3.4	70	95	yes	0	20	55.5
24	1034.6	7.1	4.9	3.1	2.2	61	96	yes	0	20	59.5
25	1032.6	10.8	7.4	4.3	3.7	46	25	no	10.1	20	28.7
26	1027.1	13.5	10.4	8.1	2.5	59	85	yes	0.4	20	21.3
27	1022.7	15.3	13	9.8	11.6	92	95	yes	0.2	20	29.6
28	1018.2	17.4	16.1	14.8	15.7	98	100	yes	0	60	28.8
29	1017.9	17.4	16.6	15.9	15.9	96	91	yes	0.2	50	25
30	1020	19.9	17.6	16.2	15.5	88	80	no	3.3	50	21.2
31	1019.9	16.2	15.7	15.3	13.4	86	93	yes	0.1	70	43.1

Figure 4.9: **Predicted Data**

In figure 4.9 shows the data that has been collected through previous data as an predicted data gathered from weather database the value shows the value of temp,pressure,mintemp,rainfall etc...

4.5.5 XGBoost Algorithm



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
import requests
import json
from sqlalchemy import create_engine
from twilio.rest import Client
agro_weather_data = pd.read_csv('/content/ml/MyDrive/ml/agro_weather_data.csv')
agro_weather_data['winddirection'].fillna(agro_weather_data['winddirection'].mean(), inplace=True)
agro_weather_data['windspeed'].fillna(agro_weather_data['windspeed'].mean(), inplace=True)
label_encoder = LabelEncoder()
agro_weather_data['rainfall'] = label_encoder.fit_transform(agro_weather_data['rainfall'])
X = agro_weather_data.drop(columns=['rainfall'])
y = agro_weather_data['rainfall']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
xgb_model = XGBClassifier(
    objective='binary:logistic',
    learning_rate=0.05,
    max_depth=6,
    n_estimators=1000,
    verbosity=0,
    early_stopping_rounds=50,
    use_label_encoder=False
)
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=50)
y_pred = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{classification_rep}")
def fetch_weather_data(api_key, location):
    url = f"https://data.api.xweather.com/conditions/{location}?format=json&plimit=1&filter=1min&client_id={api_key}&client_secret={api_key}"
    response = requests.get(url)
    data = json.loads(response.content)
    return data
engine = create_engine("mysql://user:password@host/dbname")
weather_data = fetch_weather_data("YOUR_API_KEY", "thiruvallur district, tamil nadu")
df = pd.DataFrame(weather_data)
df.to_sql("weather_data", con=engine, if_exists="replace", index=False)
def get_advisory(weather_data, crop_type):
    if weather_data.get("temperature", 0) > 25 and crop_type == "rice":
        return "Irrigate the field to prevent water stress."
    elif weather_data.get("humidity", 0) > 60 and crop_type == "wheat":
        return "Apply fungicides to prevent fungal diseases."
    else:
        return "No advisory available."
def send_sms_alert(farmer_phone_number, advisory):
```

Figure 4.10: **XGBoost Algorithm**

In figure 4.10 depicts the classification and regression part of the predicted values given can solve through this algorithm to show better results on accuracy,recall,f1-score,precision

4.5.6 Module3: User Interface and Notification System

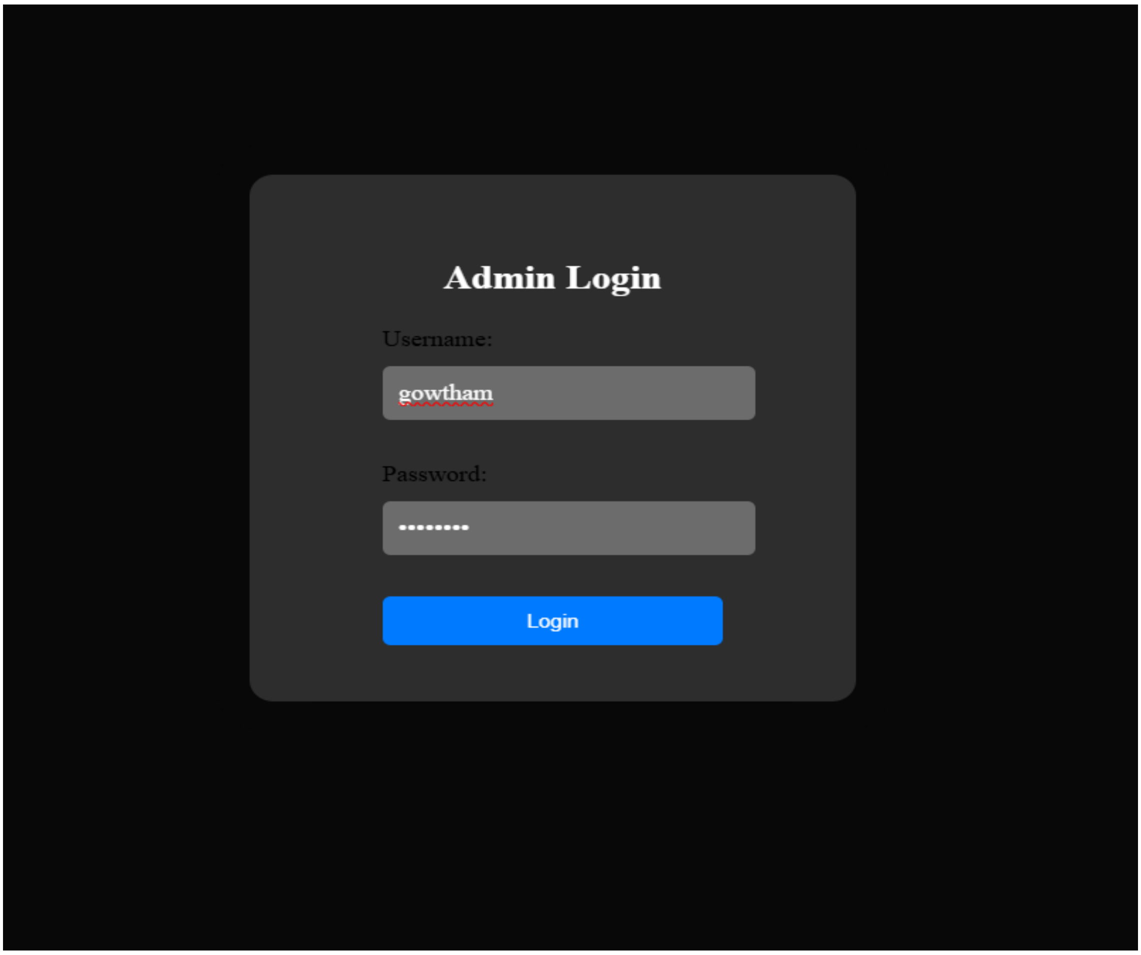
This module provides the user interface, which allows farmers to access weather information and crop advisories. The system also includes SMS notifications to alert users about critical weather changes.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and output

5.1.1 Input Design



Admin Login

Username:

gowtham

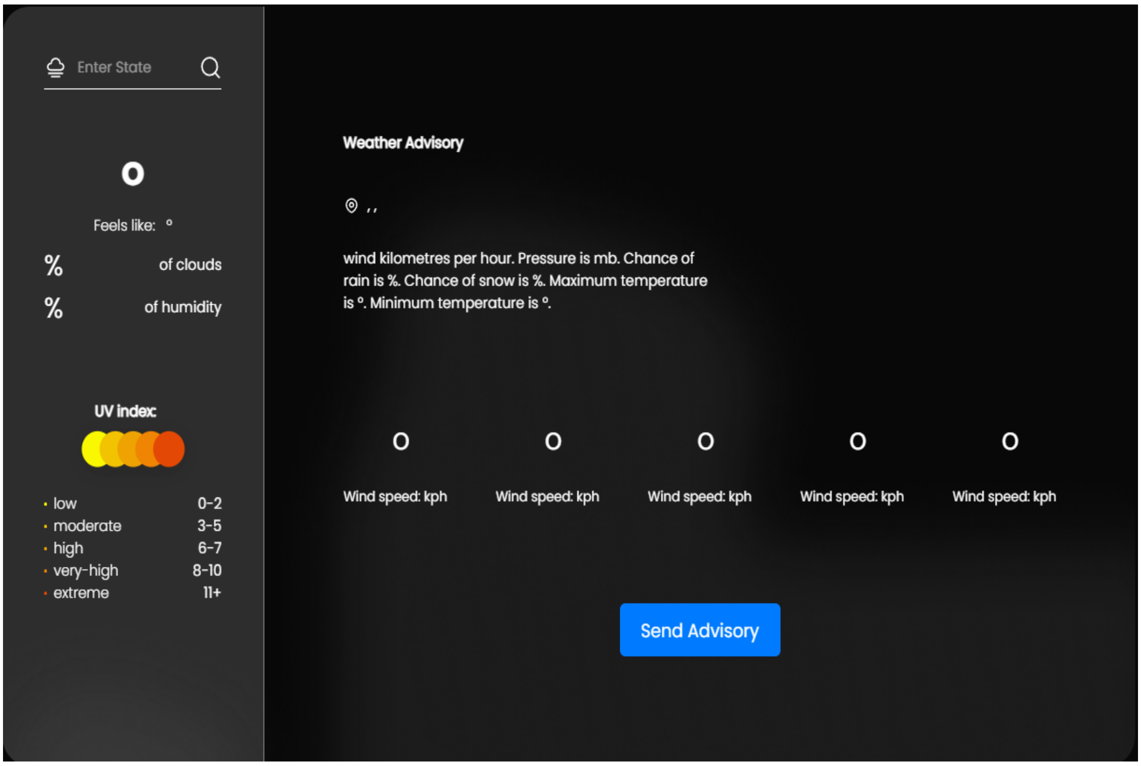
Password:

••••••••

Figure 5.1: **Input Design**

This Figure 5.1 shows the input page for weather forecast where the admin can only access through the web page for login purpose their data has been registered and they were allowed to run the page

5.1.2 output Design



wind kilometres per hour. Pressure is mb. Chance of rain is %. Chance of snow is %. Maximum temperature is °. Minimum temperature is °.

Figure 5.2: **output Design**

This Figure 5.2 shows the output page where the admin can access through the web page for sending the advisory for the farmers who have been registered

5.2 Testing

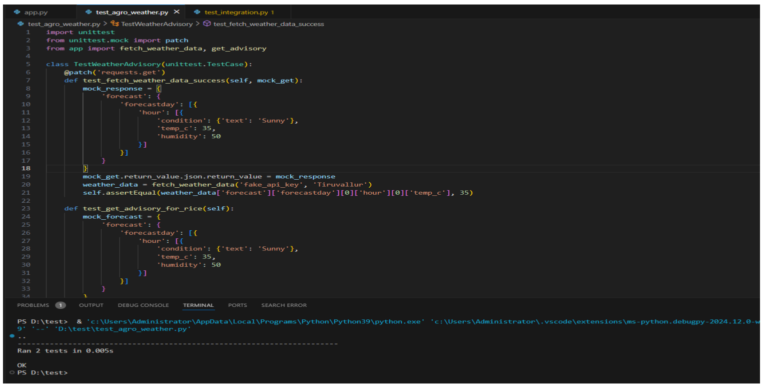
5.3 Types of Testing

5.3.1 Unit testing

Input

```
import unittest
from unittest.mock import patch
from app import fetch_weather_data , get_advisory

class TestWeatherAdvisory(unittest.TestCase):
    @patch('requests.get')
    def test_fetch_weather_data_success(self, mock_get):
        mock_response = {
            'forecast': {
                'forecastday': [{
                    'hour': [{
                        ’condition’: {’text’: ’Sunny’},
                        ’temp_c’: 35,
                        ’humidity’: 50
                    }]
                }]
            }
        }
        mock_get.return_value.json.return_value = mock_response
        weather_data = fetch_weather_data(’fake_api_key’, ’Tiruvallur’)
        self.assertEqual(weather_data[’forecast’][’forecastday’][0][’hour’][0][’temp_c’], 35)

    def test_get_advisory_for_rice(self):
        mock_forecast = {
            ’forecast’: {
                ’forecastday’: [{
                    ’hour’: [{
                        ’condition’: {’text’: ’Sunny’},
                        ’temp_c’: 35,
                        ’humidity’: 50
                    }]
                }]
            }
        }
        advisory_message = get_advisory(mock_forecast, ”Rice”, 0, 75)
        self.assertIn(”Irrigate the field to prevent water stress.”, advisory_message)

if _name_ == ’_main_’:
    unittest.main(exit=False)
```

Test result



```
import unittest
from unittest.mock import patch
from app import fetch_weather_data, get_advisory

class TestWeatherAdvisory(unittest.TestCase):
    @patch('requests.get')
    def test_fetch_weather_data_success(self, mock_get):
        mock_response = {
            'forecast': {
                'forecastday': [{
                    'hour': [{
                        'condition': {'text': 'Sunny'},
                        'temp_c': 35,
                        'humidity': 50
                    }]
                }]
            }
        }
        mock_get.return_value.json.return_value = mock_response
        weather_data = fetch_weather_data('fake_api_key', 'Tiruvallur')
        self.assertEqual(weather_data['forecast']['forecastday'][0]['hour'][0]['temp_c'], 35)

    def test_get_advisory_for_rice(self):
        mock_forecast = {
            'forecast': {
                'forecastday': [{
                    'hour': [{
                        'condition': {'text': 'Sunny'},
                        'temp_c': 35,
                        'humidity': 50
                    }]
                }]
            }
        }
```

```
PS D:\test>  & 'c:\Users\Administrator\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.debugpy-2024.12.0-w
9' '--' 'D:\test\test_agro_weather.py'
..
----------------------------------------------------------------------
Ran 2 tests in 0.005s

OK
PS D:\test>
```

Figure 5.3: **Unit Testing output**

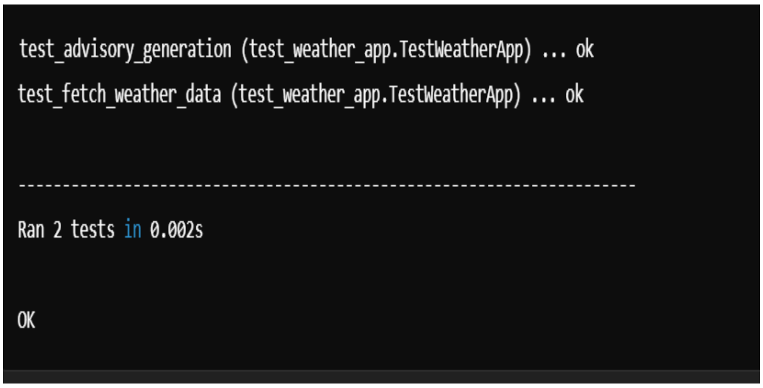
This figure 5.3 demonstrates the structure and workflow of the unit testing framework designed for agro-weather application code. The framework ensures accuracy and reliability in delivering weather information for agricultural purposes.

5.3.2 Integration testing

Input

```
import unittest
from app import app

class FlaskTestCase(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True

    def test_api_weather_fetch(self):
        result = self.app.get('/api/weather?city=Tiruvallur')
        self.assertEqual(result.status_code, 200)
        self.assertIn('weather', result.json)

    def test_advisory_generation(self):
        result = self.app.get('/api/advisory?crop_type=Rice&city=Tiruvallur')
        self.assertEqual(result.status_code, 200)
        self.assertIn('advisory', result.json)

if _name_ == '_main_':
    unittest.main()
```

Test result



```
test_advisory_generation (test_weather_app.TestWeatherApp) ... ok
test_fetch_weather_data (test_weather_app.TestWeatherApp) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```

Figure 5.4: **Integration Testing output**

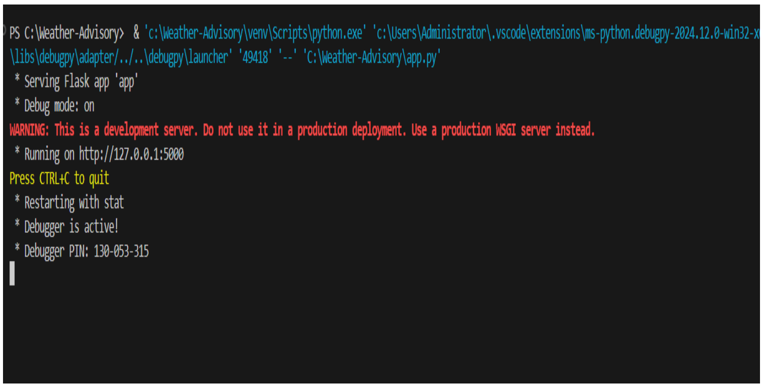
This figure 5.4 represents the integration testing framework for an agro-weather agriculture system, ensuring that individual modules work cohesively to deliver accurate, actionable weather information for agricultural decision-making.

5.3.3 System testing

Input

```
import pandas as pd
from flask import Flask, render_template, jsonify, request
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier

app = Flask(_name_)

def send_sms(mobile_numbers, message):
    client = Client(TWILIo_ACCoUNT_SID, TWILIo_AUTH_ToKEN)
    for mobile_number in mobile_numbers:
        try:
            message_response = client.messages.create(
                body=message,
                from_=TWILIo_PHoNE_NUMBER,
                to=mobile_number
            )
            print(f”Message sent to {mobile_number}: {message_response.sid}”)
        except Exception as e:
            print(f”Failed to send message to {mobile_number}: {e}”)

engine = create_engine(r”mysql+pymysql://root:1234@localhost/farmers_data”)

def get_farmers_data():
    query = ”SELECT mobile_number, crops_cultivated FRoM farmers”
    farmers_df = pd.read_sql(query, engine)
    return farmers_df

if _name_ == '_main_':
    weather_data = fetch_weather_data(WEATHER_API_KEY, LoCATIoN)

    if weather_data:
        farmers_df = get_farmers_data()
        specific_hour = 3

        for i, row in farmers_df.iterrows():
            mobile_number = row['mobile_number']
            crop_type = row['crops_cultivated']
            rainfall_prediction_percentage = rainfall_percentage.mean()
            advisory_message = get_advisory(weather_data, crop_type, specific_hour,
                rainfall_prediction_percentage)
            print(f”Sent Advisory to {mobile_number}: {advisory_message}”)
            send_sms([mobile_number], advisory_message)
if _name_ == '_main_':
                app.run(debug=True)
```

Test Result



```
PS C:\Weather-Advisory> & 'c:\Weather-Advisory\venv\Scripts\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.debugpy-2024.12.0-win32-x
\libs\debugpy\adapter/../..\debugpy\launcher' '49418' '--' 'C:\Weather-Advisory\app.py'
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 130-053-315
```

Figure 5.5: **System Testing output**

This figure 5.5 illustrates the system testing framework for an agro-weather agriculture application, focusing on end-to-end validation to ensure the entire system functions as intended in real-world scenarios. It highlights the flow from data intake to user-facing outputs, verifying overall functionality, reliability, and performance

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed Agro-weather information system utilizes advanced machine learning models, specifically the XGBoost algorithm, to provide precise weather predictions and agricultural advisories. The system's efficiency is evident in its ability to predict rainfall, temperature variations, and other crucial weather parameters with an accuracy ranging from 80% to 85%. This high accuracy is achieved through the use of XGBoost, which outperforms traditional models such as decision trees due to its capability to handle large datasets, manage missing values effectively, and implement regularization to prevent overfitting.

The XGBoost algorithm builds upon decision tree models by using an ensemble approach, where multiple weak learners (decision trees) are boosted and combined to form a strong learner. It does this in two phases:

Training Phase: The XGBoost algorithm first extracts subsamples from the original dataset using the bootstrap resampling method, which generates multiple training datasets. For each of these datasets, decision trees are built, which together form the XGBoost model.

Classification Phase: In this phase, the model aggregates the outputs of the individual decision trees and uses a majority voting system to classify the results, thus enhancing accuracy. This step ensures that the predictions are robust and generalized, as the algorithm avoids overfitting and accounts for various data patterns.

Additionally, the system incorporates real-time weather data into its advisory generation, improving decision-making support for farmers by providing timely recommendations. As a result, farmers can optimize their agricultural practices, such as crop irrigation, pest management, and planting schedules, based on accurate forecasts.

6.2 Comparison of Existing and Proposed System

Existing system:(Decision tree)

The proposed system leverages machine learning models like XGBoost and LightGBM to predict weather patterns and offer agricultural advisories. These algorithms have been shown to improve accuracy significantly over traditional systems. The XGBoost model works by boosting decision trees and has an accuracy ranging from 80% to 85%, which is higher compared to basic decision trees. This improvement is due to its capability of handling large datasets with missing values, its regularization techniques, and the ability to process data efficiently. Additionally, LightGBM, which processes large amounts of data at high speed, ensures that the predictions are accurate and timely, crucial for the farmers' decision-making processes.

By integrating both historical and real-time weather data, the system helps farmers optimize their crop planting, irrigation, and pest control practices. Compared to traditional decision-tree-based systems, the proposed model is more efficient, with a higher accuracy of weather forecasts. This ultimately leads to better decision-making support for farmers, improving agricultural productivity.

Proposed system:(XGBoost algorithm)

The XGBoost algorithm reduces overfitting by using regularization techniques and allows for the integration of diverse weather datasets. This ensures that the predictions are not only more accurate but also timely, enabling farmers to make informed decisions about irrigation, pest control, and crop planting.

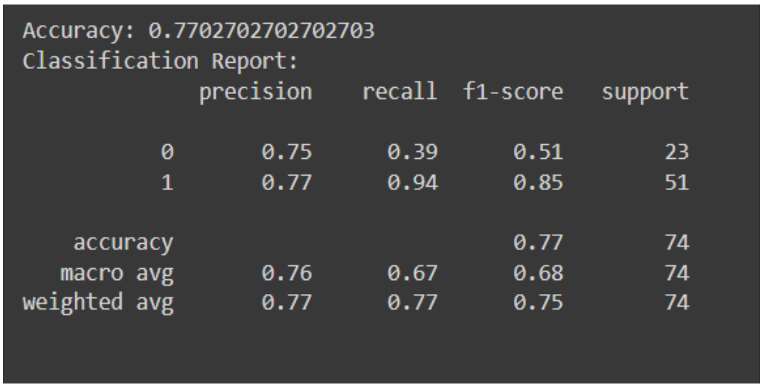
In addition, the LightGBM algorithm, which is also used in the system, enhances the overall performance by processing large datasets quickly and efficiently. Its high-speed computation is crucial for delivering real-time weather predictions and agricultural advisories, which are vital for farmers in making time-sensitive decisions.

overall, the proposed system outperforms traditional decision-tree-based models by offering higher accuracy, adaptability, and robustness. This leads to improved agricultural productivity and sustainability, helping farmers better manage resources and achieve optimal yields. The integration of real-time and historical weather data in conjunction with machine learning models such as XGBoost and LightGBM ensures that the system remains efficient and reliable in providing actionable information to farmers.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
import requests
import json
from sqlalchemy import create_engine
from twilio.rest import Client
agro_weather_data = pd.read_csv(’/content/ml/MyDrive/ml/agro_weather_data.csv’)
agro_weather_data[’winddirection’].fillna(agro_weather_data[’winddirection’].mean(), inplace=True)
agro_weather_data[’windspeed’].fillna(agro_weather_data[’windspeed’].mean(), inplace=True)
label_encoder = LabelEncoder()
agro_weather_data[’rainfall’] = label_encoder.fit_transform(agro_weather_data[’rainfall’])
X = agro_weather_data.drop(columns=[’rainfall’])
y = agro_weather_data[’rainfall’]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
xgb_model = XGBClassifier(
    objective=’binary:logistic’,
    learning_rate=0.05,
    max_depth=6,
    n_estimators=1000,
    verbosity=0,
    early_stopping_rounds=50,
    use_label_encoder=False
)
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=50)
y_pred = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(f”Accuracy: {accuracy}”)
print(f”Classification Report:\n{classification_rep}”)
def fetch_weather_data(api_key, location):
    url = f”https://data.api.xweather.com/conditions/{location}?format=json&plimit=1&filter=1min&
        client_id={api_key}&client_secret={api_key}”
    response = requests.get(url)
    data = json.loads(response.content)
    return data
engine = create_engine(”mysql://user:password@host/dbname”)
weather_data = fetch_weather_data(”YoUR_API_KEY”, ”thiruvallur district, tamil nadu”)
df = pd.DataFrame(weather_data)
df.to_sql(”weather_data”, con=engine, if_exists=”replace”, index=False)
def get_advisory(weather_data, crop_type):
    if weather_data.get(”temperature”, 0) > 25 and crop_type == ”rice”:
        return ”Irrigate the field to prevent water stress.”
    elif weather_data.get(”humidity”, 0) > 60 and crop_type == ”wheat”:
        return ”Apply fungicides to prevent fungal diseases.”
    else:
        return ”No advisory available.”
```

```
def send_sms_alert(farmer_phone_number, advisory):
    account_sid = ”YoUR_TWILIo_ACCoUNT_SID”
    auth_token = ”YoUR_TWILIo_AUTH_ToKEN”
    client = Client(account_sid, auth_token)
    message = client.messages.create(
        body=advisory,
        from_=”+14158739895”,
        to=farmer_phone_number
    )
    print(message.sid)
weather_data = fetch_weather_data(”YoUR_API_KEY”, ”thiruvallur district, tamil nadu”)
advisory = get_advisory(weather_data, ”rice”)
send_sms_alert(”FARMER_PHoNE_NUMBER”, advisory)
```

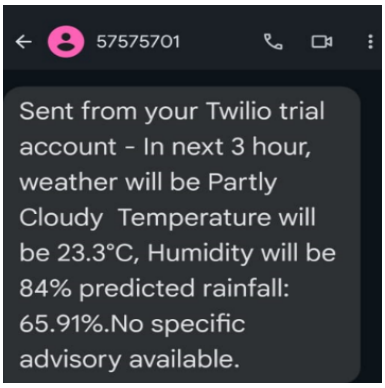
output



```
Accuracy: 0.7702702702702703
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.39      0.51        23
           1       0.77      0.94      0.85        51

    accuracy                           0.77        74
   macro avg       0.76      0.67      0.68        74
weighted avg       0.77      0.77      0.75        74
```

Figure 6.1: **output 1**

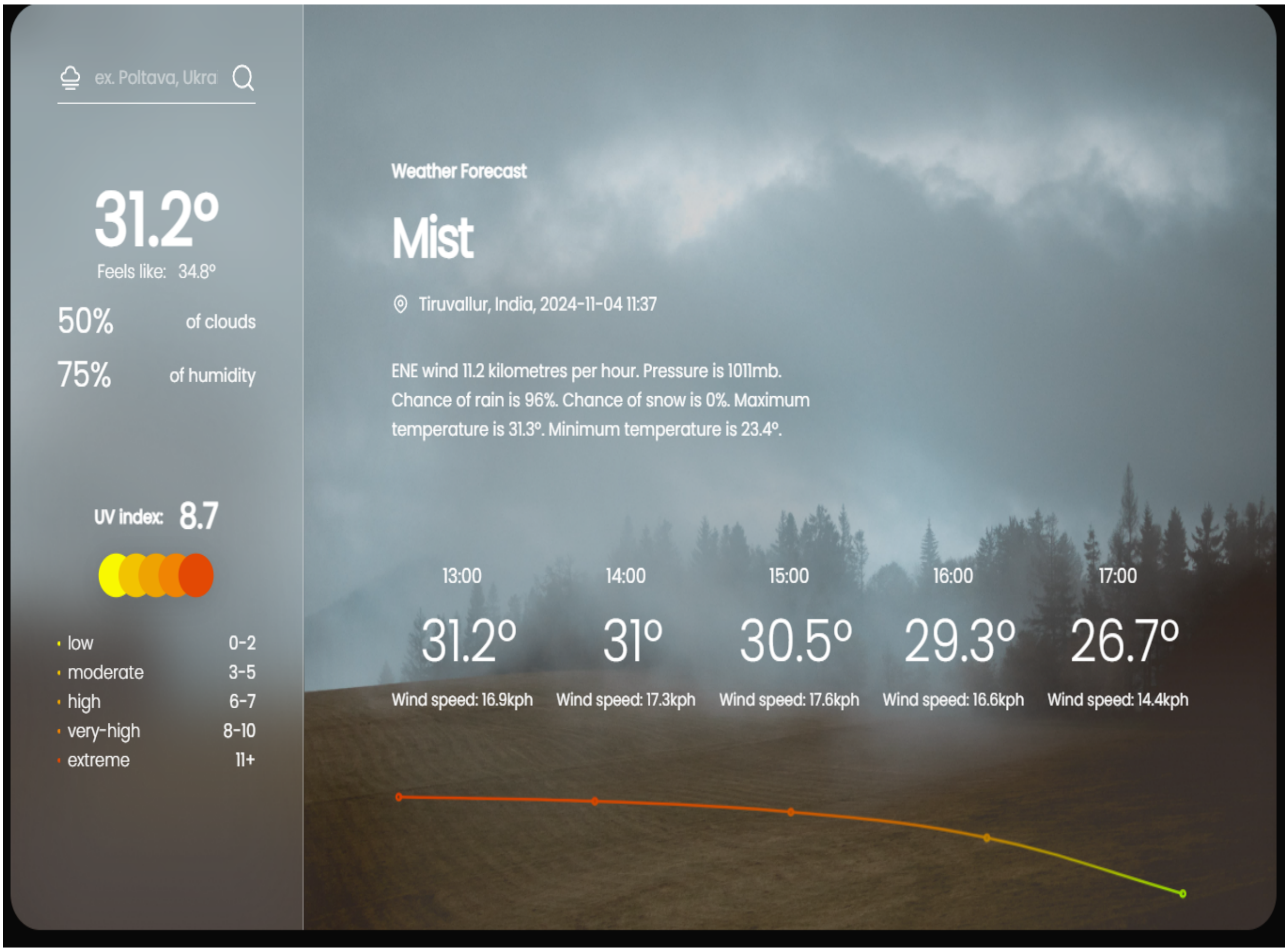
In figure 6.1 shows the output value that has been predicted using the XGBoost algorithm that shows precision,recall,f1-score,support.



Sent from your Twilio trial account - In next 3 hour, weather will be Partly Cloudy Temperature will be 23.3°C, Humidity will be 84% predicted rainfall: 65.91%.No specific advisory available.

Figure 6.2: **output 2**

In figure 6.2 shows the weather advisory for the given number that has been registered through their location,name they will be getting a message regarding the weather



31.2°
Feels like: 34.8°
50% of clouds
75% of humidity

UV index: 8.7

low	0-2
moderate	3-5
high	6-7
very-high	8-10
extreme	11+

Weather Forecast

Mist

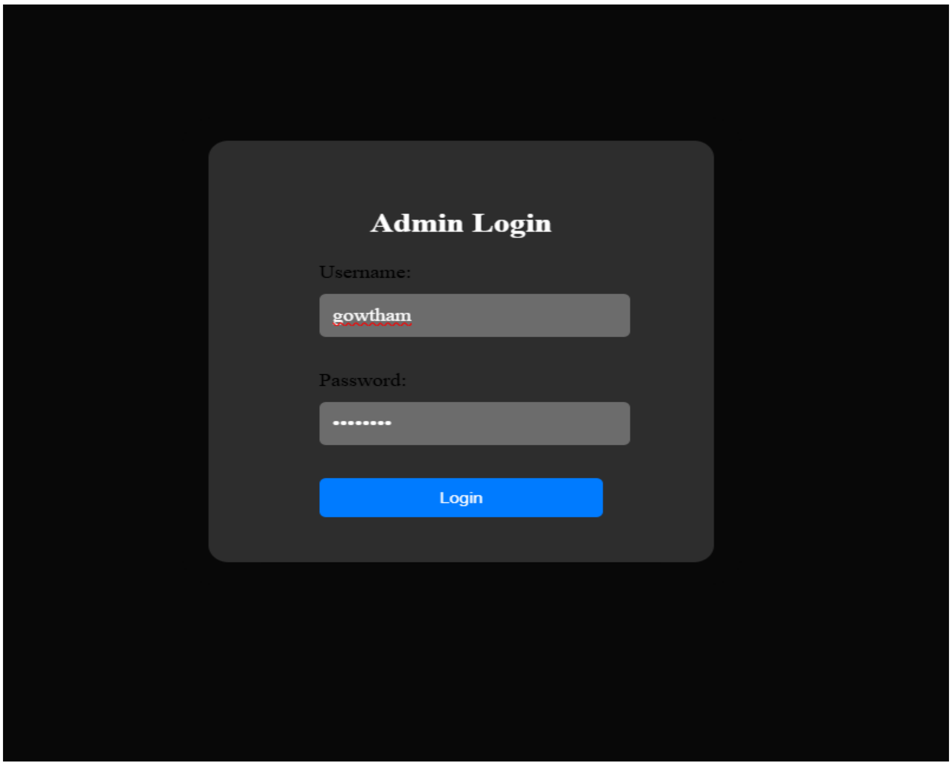
Tiruvallur, India, 2024-11-04 11:37

ENE wind 11.2 kilometres per hour. Pressure is 1011mb. Chance of rain is 96%. Chance of snow is 0%. Maximum temperature is 31.3°. Minimum temperature is 23.4°.

13:00	14:00	15:00	16:00	17:00
31.2°	31°	30.5°	29.3°	26.7°
Wind speed: 16.9kph	Wind speed: 17.3kph	Wind speed: 17.6kph	Wind speed: 16.6kph	Wind speed: 14.4kph

Figure 6.3: **web page**

In figure 6.3 shows the webpage that shows the weather forecast and temperature for upcoming hours,days therefore we can search through the location where we needed the see the details



Admin Login

Username:

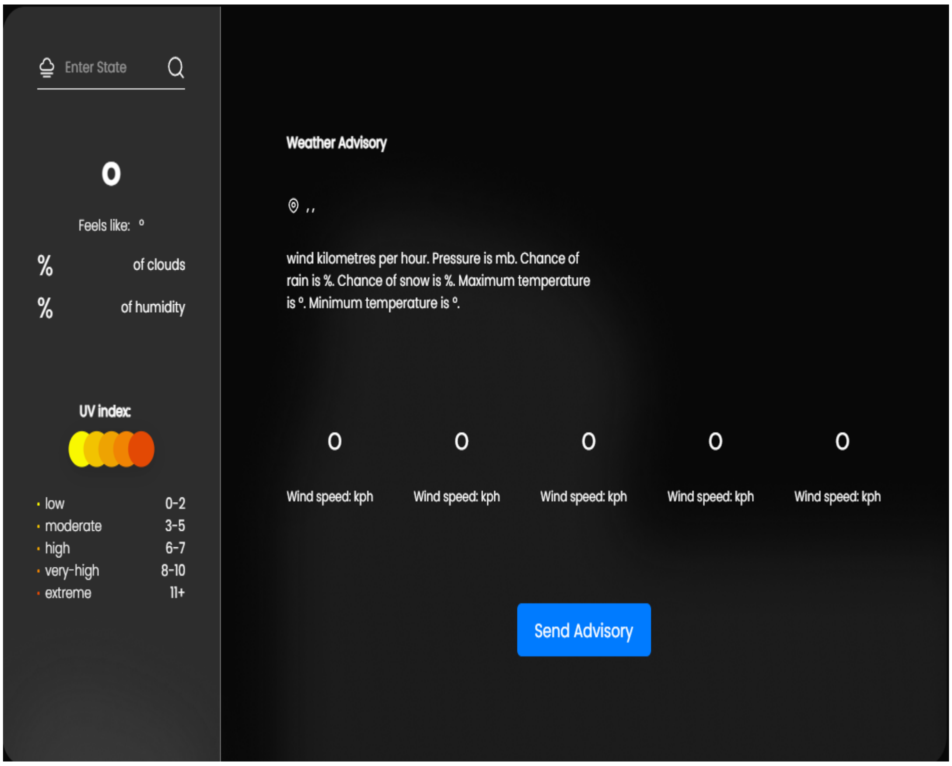
gowtham

Password:

•••••••••

Figure 6.4: **Admin Login page**

This Figure 6.4 shows the input page for weather forecast where the admin can only access through the web page for login purpose their data has been registered and they were allowed to run the page



wind kilometres per hour. Pressure is mb. Chance of rain is %. Chance of snow is %. Maximum temperature is °. Minimum temperature is °.

• low	0-2
• moderate	3-5
• high	6-7
• very-high	8-10
• extreme	11+

Figure 6.5: **SMS page**

This Figure 6.5 shows the output page where the admin can access through the web page for sending the advisory for the farmers who have been registered

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

In conclusion, the reviewed studies collectively underscore the transformative potential of advanced machine learning techniques and optimization methods in improving weather prediction accuracy. Significant progress has been achieved through the successful application of models like Extreme Gradient Boosting (XGBoost) in various meteorological domains, which our Agro Weather Project leverages. These approaches have demonstrated enhanced prediction capabilities, particularly in forecasting short-term weather patterns and rainfall, crucial for agriculture.

However, despite these advancements, the existing research reveals critical gaps that warrant further investigation. One key challenge is the generalizability of these models across diverse climatic regions. While XGBoost and similar algorithms perform well in specific regions with ample data, their effectiveness can diminish in areas where data is sparse or highly variable. Our project aims to address this by incorporating data from rural areas like Aranvoyal, but future research should focus on refining models to better adapt to different climates and microclimates, ensuring that they remain accurate across a wider geographic range.

Moreover, the impact of varying data sources, including satellite data, sensor networks, and historical meteorological records, along with seasonal and cyclical factors, remains underexplored. Seasonal variations, such as monsoons, droughts, or cold waves, introduce complexities that current models may struggle to account for. Integrating these diverse data sources in a more comprehensive way, perhaps using ensemble methods, could enhance prediction accuracy in such dynamic environments.

The integration of multiple predictive approaches, such as combining machine learning with physical models or using hybrid AI methods, offers a promising avenue for more robust and reliable solutions. Such an approach would allow systems like our Agro Weather Project to adapt to real-time conditions, detect anomalies, and respond more effectively to extreme weather events like floods, storms, or prolonged droughts. Additionally, the inclusion of real-time monitoring through IoT devices could further improve the responsiveness of weather advisory systems.

Another important area for future research is ensuring the long-term sustainability and reliability of these predictions in the face of climate change. As climate patterns become increasingly erratic, it is vital that prediction models not only account for historical trends but also incorporate future climate projections. This could help in developing adaptive systems that are resilient to the shifting baseline of global weather patterns.

By addressing these critical gaps, future research can contribute to more reliable and actionable weather forecasts. For agricultural sectors, improved forecasts mean better decision-making for planting, irrigation, pest management, and harvesting, ultimately enhancing productivity and sustainability. Furthermore, more accurate weather prediction systems will be pivotal in improving preparedness for weather-related challenges, aiding sectors like disaster management, public health, and infrastructure planning. Our Agro Weather Project aims to contribute to this field by delivering localized, precise weather data and crop-specific advisories to rural farmers, helping them make informed decisions that improve agricultural outcomes.

7.2 Future Enhancements

one key improvement is the integration of machine learning algorithms to refine weather predictions and crop advisories further. By analyzing historical weather patterns alongside real-time data, the system can provide increasingly accurate forecasts and personalized recommendations tailored to individual farms. Incorporating data from drone technology could enhance the system's ability to monitor crop health, soil conditions, and pest populations, offering farmers real-time insights and actionable data.

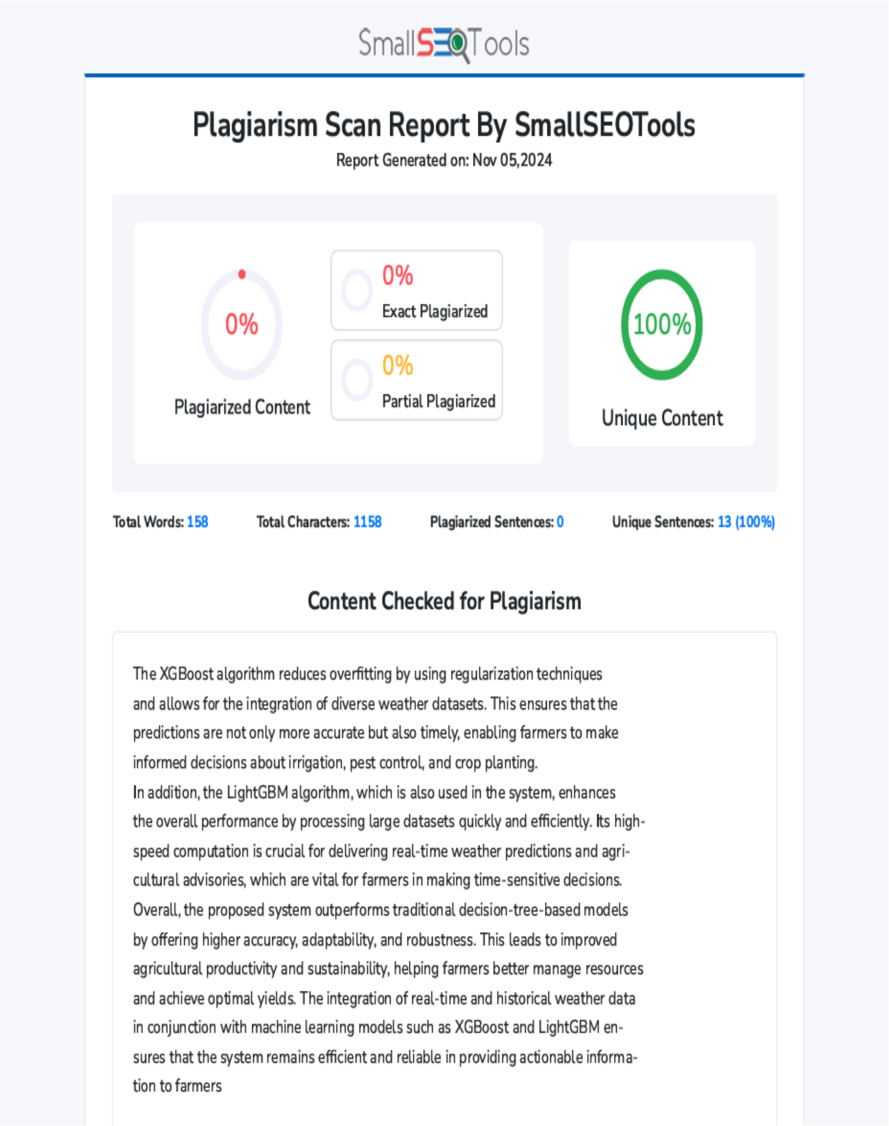
Another enhancement could involve developing a community feedback feature, where farmers can share their experiences and outcomes related to the system's recommendations. This could help improve the quality of data used for predictions and create a knowledge-sharing platform among farmers, promoting a collaborative farming community.Create forums or chat functions within the app for farmers to share experiences, challenges, and solutions. This could encourage a sense of community and collective problem-solving.

Additionally, expanding the system's language options and enhance accessibility, ensuring that more farmers can benefit from its services. Incorporating an educational component within the app could provide farmers with training on sustainable practices, pest management, and climate resilience strategies, empowering them with knowledge alongside actionable advice.

Finally,Automated Alerts and Notifications to Implement automated alerts for extreme weather conditions, market price changes, and pest outbreaks, ensuring farmers receive timely notifications via SMS, app, or other communication channels.

Chapter 8

PLAGIARISM REPORT



Plagiarism Scan Report By SmallSEOTools

Report Generated on: Nov 05,2024

Total Words: 158 Total Characters: 1158 Plagiarized Sentences: 0 Unique Sentences: 13 (100%)

Content Checked for Plagiarism

The XGBoost algorithm reduces overfitting by using regularization techniques and allows for the integration of diverse weather datasets. This ensures that the predictions are not only more accurate but also timely, enabling farmers to make informed decisions about irrigation, pest control, and crop planting.

In addition, the LightGBM algorithm, which is also used in the system, enhances the overall performance by processing large datasets quickly and efficiently. Its high-speed computation is crucial for delivering real-time weather predictions and agricultural advisories, which are vital for farmers in making time-sensitive decisions. Overall, the proposed system outperforms traditional decision-tree-based models by offering higher accuracy, adaptability, and robustness. This leads to improved agricultural productivity and sustainability, helping farmers better manage resources and achieve optimal yields. The integration of real-time and historical weather data in conjunction with machine learning models such as XGBoost and LightGBM ensures that the system remains efficient and reliable in providing actionable information to farmers

Figure 8.1: **Plagiarism Report**

In figure 8.1 shows the plagiarism report for the project of agro weather information has been shown 0 plagiarism content for the content and shown as an unique content.

Appendices

Appendix A

Complete Data / Sample Data / Sample Source Code / etc

A.1 A complete Data

A.1.1 Dataset overview

The dataset for agro weather agriculture includes several key columns that provide valuable insights into farming practices and weather conditions. The Name column captures the name of the farmer or agricultural worker, while the Mobile Number column allows for easy communication, formatted to include the country code. The Location column specifies the geographical area of the farm, which can be presented as a city, state, or even GPS coordinates. The Crops Cultivated column lists the types of crops grown, which could be represented as a string or a comma-separated list. Weather conditions are documented in the Reported Weather column, summarizing factors such as temperature and rainfall, potentially in a structured format like JSoN. Finally, the Crop Duration column indicates the time needed for each crop to mature, expressed in days or months. This structured dataset not only facilitates the analysis of the relationship between agricultural practices and weather patterns but also emphasizes the importance of data privacy and quality management in agricultural research.

A.1.2 Dataset categories

- Name: The name of the farmer or contact person.
- Mobile Number: The primary contact number for the farmer.
- Location: The geographical location, such as village or district, where the crops are grown.
- Crops Cultivated: Types of crops currently cultivated (e.g., rice, wheat, maize).

- Reported Weather: Current weather conditions reported by the farmer or recorded (e.g., rainy, dry, humid).
- Crop Duration: Expected or actual growth period of each crop, helping to track seasonal cycles and harvest times (e.g., 4-6 months).

A.2 Sample Data

Name	Mobile No	Location	Crops Cultivated	Reported Weather
Ramasamy	9876543210	Aranvoyal	Rice, Tomatoes, Mangoes	mist
Meenakshi	8765432109	Aranvoyal	Sugarcane, Okra, Bananas	Light Rain
Murugan	7654321098	Aranvoyal	Millets, Brinjal, Mangoes	Sunny
Lakshmi	7896541230	Aranvoyal	Vegetables, Finger Millet, Bananas	Cloudy
Perumal	8976543211	Aranvoyal	Sugarcane, Tomatoes, Bananas	Heavy Rain
Saraswathi	9876543212	Aranvoyal	Groundnut, Okra, Mangoes	Light Rain
Gopal	8765432108	Aranvoyal	Pulses, Brinjal, Finger Millet	Moderate Rain
Kamala	7654321097	Aranvoyal	Rice, Vegetables, Bananas	Sunny
Rajan	7896541231	Aranvoyal	Vegetables, Groundnut, Mangoes	Cloudy
Vijaya	8976543213	Aranvoyal	Sugarcane, Pulses, Okra	Heavy Rain

Table A.2: sample data

A.3 Sample Source Code

A.3.1 Python Code

```

    app.py:

import pandas as pd
from flask import Flask, render_template, jsonify, request, redirect, url_for, flash
from flask_cors import CoRS
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import requests
import json
from sqlalchemy import create_engine
from twilio.rest import Client
import os

app = Flask(__name__)
CoRS(app)

app.secret_key = 'your_secret_key'
admin_username = 'admin'
admin_password = 'password'

class Config:
    WEATHER_API_KEY = os.environ.get("WEATHER_API_KEY")
    WEATHER_API_URL = 'http://api.weatherapi.com/v1/forecast.json'
    TWILIo_ACCoUNT_SID = os.environ.get("TWILIo_ACCoUNT_SID")
    TWILIo_AUTH_ToKEN = os.environ.get("TWILIo_AUTH_ToKEN")
    TWILIo_PHoNE_NUMBER = os.environ.get("TWILIo_PHoNE_NUMBER")
    LoCATIoN = "Tiruvallur"

config = Config()

@app.route('/',)
def index():
    return render_template('index.html')
import pandas as pd
from flask import Flask, render_template, jsonify, request,redirect,url_for,flash
from flask_cors import CoRS
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import requests
import json
```

```
from sqlalchemy import create_engine
from twilio.rest import Client
import os

app = Flask(__name__)
CoRS(app)

@app.route('/')
def home():
    return render_template('index.html')

app.secret_key = 'your_secret_key'

admin_username = 'admin'
admin_password = 'password'

@app.route('/admin-login', methods=['GET', 'PoST'])
def admin_login():
    if request.method == 'PoST':
        username = request.form['username']
        password = request.form['password']

        if username == admin_username and password == admin_password:
            return redirect(url_for('admin_dashboard'))
        else:
            flash('Invalid username or password')
            return render_template('login.html')

    return render_template('login.html')
@app.route('/admin-dashboard')
def admin_dashboard():
    return render_template("admin_dashboard.html")

WEATHER_API_KEY = '563b8d6c849242fb82c135303240111'
WEATHER_API_URL = 'http://api.weatherapi.com/v1/forecast.json?key=563
    b8d6c849242fb82c135303240111&q=tiruvallur&days=1&aqi=yes&alerts=yes'
TWILIo_ACCoUNT_SID = 'ACbcad29883125ea74cbc351047c34229e'
TWILIo_AUTH_ToKEN = '43f041fd12478fee2a80c50c5b39a1be'
TWILIo_PHoNE_NUMBER = '+14158739895'
LoCATIoN = "Tiruvallur"

agro_weather_data = pd.read_csv(r"C:\Weather-Advisory\agro_weather_data.csv")

agro_weather_data['winddirection'].fillna(agro_weather_data['winddirection'].mean())
agro_weather_data['windspeed'].fillna(agro_weather_data['windspeed'].mean())

label_encoder = LabelEncoder()
agro_weather_data['rainfall'] = label_encoder.fit_transform(agro_weather_data['rainfall'])

X = agro_weather_data.drop(columns=['rainfall'])
```

```
y = agro_weather_data['rainfall']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

xgb_model = XGBClassifier(
    objective='binary:logistic',
    learning_rate=0.05,
    max_depth=6,
    n_estimators=1000,
    verbosity=0,
    early_stopping_rounds=50
)
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=False)

y_pred_proba = xgb_model.predict_proba(X_test)

rainfall_percentage = y_pred_proba[:, 1] * 100

def fetch_weather_data(WEATHER_API_KEY, location):
    url = f"http://api.weatherapi.com/v1/forecast.json?key={WEATHER_API_KEY}&q={location}&days=1&aqi=yes&alerts=yes"
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        return data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching weather data: {e}")
        return None

def get_advisory(forecast_data, crop_type, specific_hour, rainfall_prediction_percentage):
    weather_condition = forecast_data['forecast']['forecastday'][0]['hour'][specific_hour]['condition']['text']
    temperature = forecast_data['forecast']['forecastday'][0]['hour'][specific_hour]['temp_c']
    humidity = forecast_data['forecast']['forecastday'][0]['hour'][specific_hour]['humidity']

advisory_message = (In the next (specific_hour hour) Temperature will be {temperature} C , Humidity will be (humidity), possibility of (weather_condition) weather condition.f predicted rainfall(rainfall_prediction_percentage:.2f))

    if crop_type == "Rice":
        if temperature > 30:
            advisory_message += "Irrigate the field to prevent water stress."
        elif humidity < 50:
            advisory_message += "Consider irrigation to maintain soil moisture."
    elif crop_type == "Tomatoes":
        if humidity > 70:
            advisory_message += "Apply fungicides to prevent blight."
    elif crop_type == "Mangoes":
        if temperature > 35:
            advisory_message += ”Ensure adequate watering to prevent fruit drop.”
    elif crop_type == ”Sugarcane”:
        if humidity < 60:
            advisory_message += ”Consider irrigation to support growth.”
    elif crop_type == ”okra”:
        if temperature > 32:
            advisory_message += ”Water the plants to promote healthy growth.”
    elif crop_type == ”Bananas”:
        if humidity < 50:
            advisory_message += ”Increase watering to prevent stress.”
    elif crop_type == ”Millets”:
        if temperature > 30 and humidity < 40:
            advisory_message += ”Provide adequate irrigation.”
    elif crop_type == ”Brinjal”:
        if humidity > 60:
            advisory_message += ”Watch for pests and apply pesticides if necessary.”
    elif crop_type == ”Vegetables”:
        if temperature > 28:
            advisory_message += ”Ensure proper watering to keep the plants hydrated.”
    elif crop_type == ”Finger Millet”:
        if humidity < 50:
            advisory_message += ”Irrigate to maintain soil moisture.”
    elif crop_type == ”Groundnut”:
        if temperature > 30:
            advisory_message += ”Water the plants to avoid stress.”
    elif crop_type == ”Pulses”:
        if humidity > 70:
            advisory_message += ”Apply fungicides to prevent diseases.”
    elif crop_type == ”Fruits”:
        if temperature > 30 and humidity < 50:
            advisory_message += ”Ensure adequate watering to support fruit development.”
    else:
        advisory_message += ”No specific advisory available.”

    return advisory_message

@app.route(’/login’)
def login():
    return render_template(’login.html’)

@app.route(’/get_weather’, methods=[’GET’])
def get_weather():
    city = request.args.get(’city’,’India’)
    weather_data = fetch_weather_data(WEATHER_API_KEY, city)
    if weather_data:
        return jsonify(weather_data)
    else:
        return jsonify({’error’: ’Could not fetch weather data.’}), 500

@app.route(’/get_weather_advisory’, methods=[’GET’])
```

```
def get_weather_advisory():
    crop_type = request.args.get(’crop_type’)
    specific_hour = int(request.args.get(’hour’, 1))
    weather_data = fetch_weather_data(WEATHER_API_KEY, LoCATIoN)
    if weather_data:
        rainfall_prediction_percentage = rainfall_percentage.mean()
        advisory_message = get_advisory(weather_data, crop_type, specific_hour)
        return jsonify({
        ’advisory’: advisory_message,
        ’temperature’: weather_data[’forecast’][’forecastday’][0][’hour’][specific_hour][’temp_c’],
        ’humidity’: weather_data[’forecast’][’forecastday’][0][’hour’][specific_hour][’humidity’],
        ’condition’: weather_data[’forecast’][’forecastday’][0][’hour’][specific_hour][’condition’][’text’]
    })
    else:
        return jsonify({’error’: ’Could not fetch weather data.’}), 500

def send_sms(mobile_numbers, message):
    client = Client(TWILIo_ACCoUNT_SID, TWILIo_AUTH_ToKEN)
    for mobile_number in mobile_numbers:
        try:
            message_response = client.messages.create(
                body=message,
                from_=TWILIo_PHoNE_NUMBER,
                to=mobile_number
            )
            print(f”Message sent to {mobile_number}: {message_response.sid}”)
        except Exception as e:
            print(f”Failed to send message to {mobile_number}: {e}”)

engine = create_engine(r”mysql+pymysql://root:1234@localhost/farmers_data”)

def get_farmers_data():
    query = ”SELECT mobile_number, crops_cultivated FRoM farmers”
    farmers_df = pd.read_sql(query, engine)
    return farmers_df

if __name__ == ’__main__’:
    weather_data = fetch_weather_data(WEATHER_API_KEY, LoCATIoN)

    if weather_data:
        farmers_df = get_farmers_data()

        specific_hour = 1

        for i, row in farmers_df.iterrows():
            mobile_number = row[’mobile_number’]
            crop_type = row[’crops_cultivated’]

            rainfall_prediction_percentage = rainfall_percentage.mean()

            advisory_message = get_advisory(weather_data, crop_type, specific_hour, rainfall_prediction_percentage)
            send_sms([mobile_number], advisory_message)

    app.run(debug=True)
```

A.3.2 HTML Code

```
index.html
<!DoCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      rel="icon"
      href="{{ url_for('static', filename='img/gh_descr/site-pic.png') }}"
      type="image/icon type"
    />
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}" />
    <title>Weather Forecast</title>
     </head>
  <body>
    <header> <!-- Added header for the login button -->
      <a href="{{ url_for('login.') }}" class="login-button">Admin Login</a>
  </header>
    <main>
      <section class="side-container">
        <div class="search-container">
          <svg width="22px" height="22px" class="weather-icon">
            <use href="{{ url_for('static', filename='img/svg.svg#icon-weather-icon') }}"></use>
          </svg>
          <input
            type="text"
            class="geo-input"
            placeholder="Enter city"
          />
          <button class="search-btn">
            <svg width="22px" height="22px">
              <use href="{{ url_for('static', filename='static/img/svg.svg#icon-search-icon') }}"></use>
            </svg>
          </button>
        </div>
        <div class="day-stats">
          <h2 class="day-stats__temperature">
            <span class="day-stats__temperature-value"></span>
          </h2>
          <h3 class="day-stats__feelslike">
            Feels like: &nbsp&nbsp<span
              class="day-stats__feelslike-value"
            ></span
            >
          </h3>
          <ul class="day-stats__conditions">
            <li>
              <p class="day-stats__conditon">
                <span class="day-stats__clouds"></span>
      </section>
      <section class="main-container">
        <h4 class="secondary-title">Weather Forecast</h4>
          <p class="weather__location-text">
            <span class="weather__location-city"></span>,
            <span class="weather__location-country"></span>,
            <span class="weather__location-date"></span>
          </p>
        </div>
        <p class="weather__primary-stats">
          <span class="weather__wind-dir"></span> wind
          <span class="weather__wind-kph"></span> kilometres per hour. Pressure
          is <span class="weather__pressure-mb"></span>mb. Chance of rain is
          <span class="weather__rain"></span> Chance of snow is
          <span class="weather__snow"></span> Maximum temperature is
          <span class="weather__max-temp"></span>. Minimum temperature is
          <span class="weather__min-temp"></span>.
        </p>

        <ul class="forecast">
          <li class="forecast-item">
            <p class="forecast__time"></p>
            <p class="forecast__temperature">
              &nbsp;<span class="forecast__temperature--value"></span>o
            </p>
            <p class="forecast__wind-text">
              Wind speed: <span class="forecast__wind-value"></span>kph
            </p>
          </li>
          <li class="forecast-item">
            <p class="forecast__time"></p>
            <p class="forecast__temperature">
              &nbsp;<span class="forecast__temperature--value"></span>o
            </p>
            <p class="forecast__wind-text">
              Wind speed: <span class="forecast__wind-value"></span>kph
            </p>
          </li>
          <li class="forecast-item">
            <p class="forecast__time"></p>
            <p class="forecast__temperature">
              &nbsp;<span class="forecast__temperature--value"></span>o
            </p>
            </p>
          </li>
            <p class="forecast__time"></p>
            <p class="forecast__temperature">
              &nbsp;<span class="forecast__temperature--value"></span>o
            </p>
            </p>
          </li>
        </ul>
      </section>
    </main>

  </body>
</html>

 login.html:

<!DoCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Login</title>
    <link rel="stylesheet" href="../static/css/login.css">
</head>
<body>
    <main class="login-container">
        <div class="side-container">
            <h2>Admin Login</h2>
            <form action="/admin_login" method="PoST">
                <div class="search-container">
                    <label for="username">Username:</label><br>
                    <input type="text" id="username" name="username" class="geo-input" required
                          placeholder="Enter Username"><br><br>

                    <label for="password">Password:</label><br>
                    <input type="password" id="password" name="password" class="geo-input"
                         required placeholder="Enter Password"><br><br>
                </div>
                <input type="submit" value="Login" class="search-btn">
            </form>
        </div>
    </main>
</body>
</html>

admin_dashboard.html:

<!DoCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" href="../static/img/gh_descr/site-pic.png" type="image/icon" />
    <link rel="stylesheet" href="../static/css/admin.css">
    <title>Weather Advisory </title>
  </head>
  <body>
    <main>
 </symbol>
            <symbol id="icon-search-icon" viewBox="0 0 32 32">
          </symbol>
          <symbol id="icon-location-icon" viewBox="0 0 32 32">
          </symbol>
        </defs>
    </svg>
      <section class="side-container">
        <div class="search-container">
          <svg width="22px" height="22px" class="weather-icon">
          </svg>
          <input
            type="text"
            class="geo-input"
            placeholder="Enter State"
          />
          <button class="search-btn"  aria-label="Search">
            <svg width="22px" height="22px">
            </svg>
          </button>
        </div>
        <div class="day-stats">
            </li>
          </ul>
        </div>
        <div class="uv-container">
          <h3 class="uv-header">
          </h3>
          <div class="uv-icon">
          </div>
```

```
<ul class="uv-stats">
  <li class="uv-stat">
    <div>
      <svg
        xmlns="http://www.w3.org/2000/svg"
        width="3"
        height="3"
        viewBox="0 0 3 3"
        fill="none"
        class="uv-stat__icon"
      >
        <circle cx="1.5" cy="1.5" r="1.5" fill="#F9F801" />
      </svg>
      <p class="uv-stat__text">low</p>
    </div>
    <p class="uv-value">0-2</p>
  </li>
  <li class="uv-stat">
    <div>
      <svg
        xmlns="http://www.w3.org/2000/svg"
        width="3"
        height="3"
        viewBox="0 0 3 3"
        fill="none"
        class="uv-stat__icon"
      >
        <circle cx="1.5" cy="1.5" r="1.5" fill="#F2C301" />
      </svg>
      <p class="uv-stat__text">moderate</p>
    </div>
    <p class="uv-value">3-5</p>
  </li>
  <li class="uv-stat">
    <div>
      <svg
        xmlns="http://www.w3.org/2000/svg"
        width="3"
        height="3"
        viewBox="0 0 3 3"
        fill="none"
        class="uv-stat__icon"
      >
        <circle cx="1.5" cy="1.5" r="1.5" fill="#EEA302" />
      </svg>
      <p class="uv-stat__text">high</p>
    </div>
    <p class="uv-value">6-7</p>
  </li>
  <li class="uv-stat">
```

```
        <div>
          <svg
            xmlns="http://www.w3.org/2000/svg"
            width="3"
            height="3"
            viewBox="0 0 3 3"
            fill="none"
            class="uv-stat__icon"
          >
            <circle cx="1.5" cy="1.5" r="1.5" fill="#F08403" />
          </svg>
          <p class="uv-stat__text">very-high</p>
        </div>
        <p class="uv-value">8-10</p>
      </li>
      <li class="uv-stat">
        <div>
          <svg
            xmlns="http://www.w3.org/2000/svg"
            width="3"
            height="3"
            viewBox="0 0 3 3"
            fill="none"
            class="uv-stat__icon"
          >
            <circle cx="1.5" cy="1.5" r="1.5" fill="#E34904" />
          </svg>
          <p class="uv-stat__text">extreme</p>
        </div>
        <p class="uv-value">11+</p>
      </li>
    </ul>
  </div>
  <img src="/static/img/side-blur.png" alt="Side Blur">
</section>
<section class="main-container">
  <h4 class="secondary-title">Weather Advisory</h4>
  <h1 class="weather__primary-title"></h1>
  <div class="weather__location">
    <svg width="15px" height="15px" class="location-icon">
      <use href="#icon-location-icon"></use>
    </svg>
    <p class="weather__location-text">
      <span class="weather__location-city"></span>,
      <span class="weather__location-country"></span>,
      <span class="weather__location-date"></span>
    </p>
  </div>
  <p class="weather__primary-stats">
    <span class="weather__wind-dir"></span> wind
```

```
      <span class=”weather__wind-kph”></span> kilometres per hour. Pressure
       is <span class=”weather__pressure-mb”></span>mb. Chance of rain is
      <span class=”weather__rain”></span>
      <span class=”weather__snow”></span>Maximum temperature is
      <span class=”weather__max-temp”></span> Minimum temperature is
      <span class=”weather__min-temp”></span>

    <ul class=”forecast”>
      <li class=”forecast-item”>
        <p class=”forecast__time”></p>
        <p class=”forecast__temperature”>
          &nbsp;<span class=”forecast__temperature--value”></span>
        </p>
        <p class=”forecast__wind-text”>
          Wind speed: <span class=”forecast__wind-value”></span>kph
        </p>
      </li>
      <li class=”forecast-item”>
        <p class=”forecast__time”></p>
        <p class=”forecast__temperature”>
          &nbsp;<span class=”forecast__temperature--value”></span>
        </p>
        <p class=”forecast__wind-text”>
          Wind speed: <span class=”forecast__wind-value”></span>kph
        </p>
      </li>
      <li class=”forecast-item”>
        <p class=”forecast__time”></p>
        <p class=”forecast__temperature”>
          &nbsp;<span class=”forecast__temperature--value”></span>
        </p>
        <p class=”forecast__wind-text”>
          Wind speed: <span class=”forecast__wind-value”></span>kph
        </p>
      </li>
      <li class=”forecast-item”>
        <p class=”forecast__time”></p>
        <p class=”forecast__temperature”>
          &nbsp;<span class=”forecast__temperature--value”></span>o
        </p>
        <p class=”forecast__wind-text”>
          Wind speed: <span class=”forecast__wind-value”></span>kph
        </p>
      </li>
      <li class=”forecast-item”>
        <p class=”forecast__time”></p>
        <p class=”forecast__temperature”>
          &nbsp;<span class=”forecast__temperature--value”></span>o
        </p>
        <p class=”forecast__wind-text”>
```

```
              Wind speed: <span class="forecast__wind-value"></span>kph
            </p>
          </li>
        </ul>
        <div class="send-advisory-container" >
          <button class="send-advisory-btn" Send Advisory </button>
        </div>
        <canvas id="chart" height="45px"></canvas>
      </section>
    </main>
    <script type="module" src="../static/js/script.js" type="module"></script>
    <script type="module" src="../static/js/weatherservice.js"></script>
    <script type="module" src="../static/js/chartservice.js"></script>
    <script type="module" src="../static/js/renderswebsite.js"></script>
   </body>
</html>
```

A.3.3 CSS Code

```

style.css:
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400&display=swap');

* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;
  font-style: normal;
  line-height: normal;
}

body {
  background-color: #080808;
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
  width: 100vw;
}

main {
  position: absolute;
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
  margin: 0 auto;
  width: fit-content;
  height: 635px;

  background-repeat: no-repeat;
  background-size: cover;

  border-radius: 25px;
  overflow: hidden;
}

main.sunny {
  background-image: url('../img/sunny.jpeg');
}
main.cloudy {
  background-image: url('../img/cloudy.jpeg');
}
main.sunny {
  background-image: url('../img/sunny.jpeg');
}
main.overcast {
  background-image: url('../img/overcast.jpeg');
}
main.mist {
  background-image: url('../img/mist.jpeg');
}
main.rain {
  background-image: url('../img/rain.jpeg');
}
main.snow {
  background-image: url('../img/snow.jpeg');
}
main.sleet {
  background-image: url('../img/sleet.jpeg');
}
main.drizzle {
  background-image: url('../img/drizzle.jpeg');
}
main.thunder,
main.thundery {
  background-image: url('../img/thunder.jpeg');
}
main.blizzard {
  background-image: url('../img/blizzard.jpeg');
}
main.fog {
  background-image: url('../img/fog.jpeg');
}
main.ice {
  background-image: url('../img/ice.jpeg');
```

```
}
main.shower {
  background-image: url('../img/shower.jpeg');
}
main.showers {
  background-image: url('../img/showers.jpeg');
}
main.clear {
  background-image: url('../img/clear.jpeg');
}

.side-container {
  display: flex;
  flex-direction: column;
  align-items: center;

  width: 250px;
  padding: 40px;
  height: 100;border-right: 1px solid rgba(255, 255, 255, 0.4;
  background: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(22.5px);
}

/* INPUT */
.search-container {
  width: 100;
  padding-bottom: 6px;
  margin-bottom: 55px;

  display: flex;
  align-items: center;
  justify-content: center;

  border-bottom: 1px solid #ffffff;
}

.geo-input {
  all: unset;
  flex: 1;
  width: 100;margin: 0 10px;color: rgba(255, 255, 255, 1;
  font-size: 13px;
}
.geo-input::placeholder {
  color: rgba(255, 255, 255, 0.5);
}

.search-btn {
  border: none;
  background: transparent;
  -webkit-appearance: none;
  height: 22px;

  cursor: pointer;
}

/* DAY STATS */
.day-stats {
  width: 100;
  display: flex;
  flex-direction: column;
  align-items: center;

  margin-bottom: 60px;
  z-index: 2;
}
.day-stats__temperature {
  color: #fff;
  font-size: 50px;
  line-height: 100;
}

.day-stats__temperature-value {
  line-height: 100;
}
.day-stats__feelslike {
  color: #fff;
  font-size: 13px;
  font-weight: 300;
  margin-bottom: 6px;
}

.day-stats__conditions {
  display: flex;
  flex-direction: column;
  align-items: center;

  list-style: none;
  width: 100;
}

.day-stats__conditions li {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;

  width: 100;
}
.day-stats__conditon {
  color: #fff;
  font-size: 24px;
  line-height: 100;
}

.day-stats__conditon-text {
  color: #fff;
  font-size: 13px;
}

/* UV CONTAINER */
.uv-container {
  width: 100;
  display: flex;
  flex-direction: column;
  align-items: center;
  z-index: 2;
}
.uv-header {
  color: #fff;
  font-size: 13px;

  display: flex;
  align-items: center;
}
.uv-header__value {
  font-size: 24px;
}
.uv-icon {
  filter: drop-shadow(0px 4px 11px rgba(0, 0, 0, 0.15));
}

.uv-stats {
  width: 100;
  list-style: none;
}
.uv-stat {
  width: 100;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

.uv-stat div {
  display: flex;
  flex-direction: row;
  align-items: center;
}
.uv-stat__icon {
  margin-right: 6px;
```

```
}
.uv-stat__text {
  color: #fff;
  font-size: 13px;
  font-weight: 200;
}

.uv-value {
  color: #fff;
  font-size: 13px;
}

.side-blur {
  position: absolute;
  bottom: 0;
  z-index: 1;
}

/* MAIN CONTAINER */
.main-container {
  position: relative;
  width: calc(1079px - 250px);
  height: 530px;

  display: flex;
  flex-direction: column;
  align-items: flex-start;

  padding: 0 75px;

  background-image: url('../img/main-blur.png');
}

.secondary-title {
  margin-bottom: 17px;
  color: #fff;
  font-size: 13px;
}
.weather__primary-title {
  margin-bottom: 16px;
  color: #fff;
  font-size: 38px;
  line-height: 100;
  letter-spacing: -1.9px;
}
.weather__location {
  display: flex;
  align-items: center;

  margin-bottom: 26px;
```

```
}

.location-icon {
  margin-right: 8px;
}
.weather__location-text {
  color: #fff;
  font-size: 13px;
}

.weather__primary-stats {
  width: 360px;
  color: #fff;
  font-size: 13px;

  margin-bottom: 80px;
}

/* FORECAST STATS */
.forecast {
  display: flex;
  justify-content: space-between;

  width: 100;

  list-style: none;
}

.forecast-item {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.forecast__time {
  color: #fff;
  font-size: 14px;
  margin-bottom: 4px;
}

.forecast__wind-text {
  color: #fff;
  font-size: 12px;
  margin-bottom: 10px;
}

.forecast__temperature {
  color: #fff;
  font-size: 40px;
  font-weight: 300;
  line-height: 100;
```

```
}

#chart {
  margin: 30px auto 30px;
  width: 100;
}

/* MEDIA */

@media screen and (max-width: 1100px) {
  main {
    width: 779px;
  }

  .side-container {
    width: fit-content;
  }

  .geo-input {
    font-size: 10px;
  }

  .day-stats__temperature {
    font-size: 40px;
  }

  .day-stats__conditon {
    font-size: 20px;
  }

  .day-stats__conditon-text {
    font-size: 10px;
  }

  .uv-stat__text {
    font-size: 11px;
  }

  .forecast__time {
    font-size: 12px;
  }

  .forecast__wind-text {
    font-size: 10px;
    text-align: center;
  }

  .forecast__temperature {
    font-size: 22px;
    margin-bottom: 10;
```

```css
  }
}

@media screen and (max-width: 800px) {
  main {
    width: 490px;
  }

  .main-container {
    width: 100;
    padding: 0 30px;
  }

  .weather__location {
    width: 200px;
    margin-bottom: 16px;
  }

  .weather__primary-stats {
    width: 200px;
    margin-bottom: 40px;
  }

  .forecast {
    flex-wrap: wrap;
  }

  .forecast__temperature {
    font-size: 14px;
  }

  .forecast__wind-text {
    font-size: 8px;
    width: 30px;
  }
}

login.css:

/* login.css */

/* General body styles */
body {
    background-color: #080808; /* Match the index page background */
    height: 100vh; /* Full height for the login page */
    display: flex;
    justify-content: center;
    align-items: center;
    margin: 0; /* Remove default margin */
}
```

```

/* Container for the login form */
.login-container {
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 20px;
    border-radius: 25px; /* Rounded corners */
    backdrop-filter: blur(22.5px); /* Optional blur effect */
}

/* Side container for the login form elements */
.side-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    width: 300px; /* Width of the login form */
    padding: 40px;
    background: rgba(255, 255, 255, 0.15); /* Semi-transparent background */
    backdrop-filter: blur(22.5px);
    border-radius: 15px; /* Rounded corners */
    box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1); /* Optional shadow */
}

/* Title styles */
h2 {
    color: #fff; /* Title color */
    margin-bottom: 20px; /* Space below title */
}

/* Input field styles */
.geo-input {
    all: unset; /* Reset default styles */
    width: 100; /* Full width inputs */
    margin: 10px 0; /* Space between inputs */
    padding: 10px;
    border-radius: 5px; /* Rounded corners */
    background-color: rgba(255, 255, 255, 0.3);
    color: #fff; /* Input text color */
}

/* Button styles */
.search-btn {
    width: 100; /* Full width button */
    padding: 10px;
    border: none;
    border-radius: 5px; /* Rounded corners */
    background-color: #007bff; /* Button background */
    color: #fff; /* Button text color */
    cursor: pointer; /* Pointer cursor on hover */
}

/* Button hover effect */
.search-btn:hover {
    background-color: #0056b3; /* Darker on hover */
}

admin.css:

@import url(’https://fonts.googleapis.com/css2?family=Poppins:wght@300;400&display=swap’);

* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
  font-family: ’Poppins’, sans-serif;
  font-style: normal;
  line-height: normal;
}

body {
  background-color: #080808;
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
  width: 100vw;
  float:left;
}

main {
  position: absolute;
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
  margin: 0 auto;
  width: fit-content;
  float:left;
  height: 635px;

  background-repeat: no-repeat;
  background-size: cover;

  border-radius: 25px;
  overflow: hidden;
}

main.sunny {
  background-image: url(”../img/sunny.jpeg”);
}
```

```
main.cloudy {
  background-image: url('../img/cloudy.jpeg');
}
main.sunny {
  background-image: url('../img/sunny.jpeg');
}
main.overcast {
  background-image: url('../img/overcast.jpeg');
}
main.mist {
  background-image: url('../img/mist.jpeg');
}
main.rain {
  background-image: url('../img/rain.jpeg');
}
main.snow {
  background-image: url('../img/snow.jpeg');
}
main.sleet {
  background-image: url('../img/sleet.jpeg');
}
main.drizzle {
  background-image: url('../img/drizzle.jpeg');
}
main.thunder,
main.thundery {
  background-image: url('../img/thunder.jpeg');
}
main.blizzard {
  background-image: url('../img/blizzard.jpeg');
}
main.fog {
  background-image: url('../img/fog.jpeg');
}
main.ice {
  background-image: url('../img/ice.jpeg');
}
main.shower {
  background-image: url('../img/shower.jpeg');
}
main.showers {
  background-image: url('../img/showers.jpeg');
}
main.clear {
  background-image: url('../img/clear.jpeg');
}

.side-container {
  display: flex;
  flex-direction: column;
  align-items: center;

  width: 250px;
  padding: 40px;
  height: 100;border-right: 1px solid rgba(255, 255, 255, 0.4;
  background: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(22.5px);
}

/* INPUT */
.search-container {
  width: 100;
  padding-bottom: 6px;
  margin-bottom: 55px;

  display: flex;
  align-items: center;
  justify-content: center;

  border-bottom: 1px solid #ffffff;
}

.geo-input {
  all: unset;
  flex: 1;
  width: 100;margin: 0 10px;color: rgba(255, 255, 255, 1;
  font-size: 13px;
}
.geo-input::placeholder {
  color: rgba(255, 255, 255, 0.5);
}

.search-btn {
  border: none;
  background: transparent;
  -webkit-appearance: none;
  height: 22px;

  cursor: pointer;
}

/* DAY STATS */
.day-stats {
  width: 100;
  display: flex;
  flex-direction: column;
  align-items: center;

  margin-bottom: 60px;
  z-index: 2;
```

```
}
.day-stats__temperature {
  color: #fff;
  font-size: 50px;
  line-height: 100;
}

.day-stats__temperature-value {
  line-height: 100;
}
.day-stats__feelslike {
  color: #fff;
  font-size: 13px;
  font-weight: 300;
  margin-bottom: 6px;
}

.day-stats__conditions {
  display: flex;
  flex-direction: column;
  align-items: center;

  list-style: none;
  width: 100;
}

.day-stats__conditions li {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;

  width: 100;
}
.day-stats__conditon {
  color: #fff;
  font-size: 24px;
  line-height: 100;
}

.day-stats__conditon-text {
  color: #fff;
  font-size: 13px;
}

/* UV CONTAINER */
.uv-container {
  width: 100;
  display: flex;
  flex-direction: column;
  align-items: center;
  z-index: 2;
}
.uv-header {
  color: #fff;
  font-size: 13px;

  display: flex;
  align-items: center;
}
.uv-header__value {
  font-size: 24px;
}
.uv-icon {
  filter: drop-shadow(0px 4px 11px rgba(0, 0, 0, 0.15));
}

.uv-stats {
  width: 100;
  list-style: none;
}
.uv-stat {
  width: 100;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

.uv-stat div {
  display: flex;
  flex-direction: row;
  align-items: center;
}
.uv-stat__icon {
  margin-right: 6px;
}
.uv-stat__text {
  color: #fff;
  font-size: 13px;
  font-weight: 200;
}

.uv-value {
  color: #fff;
  font-size: 13px;
}

.side-blur {
  position: absolute;
  bottom: 0;
  z-index: 1;
}

/* MAIN CONTAINER */
.main-container {
  position: relative;
  width: calc(1079px - 250px);
  height: 530px;

  display: flex;
  flex-direction: column;
  align-items: flex-start;

  padding: 0 75px;

  background-image: url('../img/main-blur.png');
}

.secondary-title {
  margin-bottom: 17px;
  color: #fff;
  font-size: 13px;
}
.weather__primary-title {
  margin-bottom: 16px;
  color: #fff;
  font-size: 38px;
  line-height: 100;
  letter-spacing: -1.9px;
}
.weather__location {
  display: flex;
  align-items: center;

  margin-bottom: 26px;
}

.location-icon {
  margin-right: 8px;
}
.weather__location-text {
  color: #fff;
  font-size: 13px;
}

.weather__primary-stats {
  width: 360px;
  color: #fff;
  font-size: 13px;

  margin-bottom: 80px;
}

/* FORECAST STATS */
.forecast {
  display: flex;
  justify-content: space-between;

  width: 100;

  list-style: none;
}

.forecast-item {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.forecast__time {
  color: #fff;
  font-size: 14px;
  margin-bottom: 4px;
}

.forecast__wind-text {
  color: #fff;
  font-size: 12px;
  margin-bottom: 10px;
}

.forecast__temperature {
  color: #fff;
  font-size: 40px;
  font-weight: 300;
  line-height: 100;
}

#chart {
  margin: 30px auto 30px;
  width: 100;
}

/* MEDIA */

@media screen and (max-width: 1100px) {
  main {
    width: 779px;
  }

  .side-container {
    width: fit-content;
  }

  .geo-input {
    font-size: 10px;
  }

  .day-stats__temperature {
    font-size: 40px;
  }

  .day-stats__conditon {
    font-size: 20px;
  }

  .day-stats__conditon-text {
    font-size: 10px;
  }

  .uv-stat__text {
    font-size: 11px;
  }

  .forecast__time {
    font-size: 12px;
  }

  .forecast__wind-text {
    font-size: 10px;
    text-align: center;
  }

  .forecast__temperature {
    font-size: 22px;
    margin-bottom: 10;
  }
}
```

A.3.4 JavaScript Code

```
 script.js:

/*
import { fetchWeatherData } from './weatherService.js';
import { createsMarkup } from './rendersWebsite.js';
import { createsChart } from './chartService.js';
*/
const refs = {
  cityInput: document.querySelector('.geo-input'),
  mainEl: document.querySelector('main'),
  searchBtn: document.querySelector('.search-btn'),
  currentTemp: document.querySelector('.day-stats__temperature-value'),
  feelslikeTemp: document.querySelector('.day-stats__feelslike-value'),
  cloudsValue: document.querySelector('.day-stats__clouds'),
  humidityValue: document.querySelector('.day-stats__humidity'),
  uvIdx: document.querySelector('.uv-header__value'),
  weatherTitle: document.querySelector('.weather__primary-title'),
  weatherCity: document.querySelector('.weather__location-city'),
  weatherCountry: document.querySelector('.weather__location-country'),
  weatherDate: document.querySelector('.weather__location-date'),
  weatherWindDir: document.querySelector('.weather__wind-dir'),
  weatherSpeed: document.querySelector('.weather__wind-kph'),
  weatherPressure: document.querySelector('.weather__pressure-mb'),
  weatherRain: document.querySelector('.weather__rain'),
  weatherSnow: document.querySelector('.weather__snow'),
  weatherMaxTemp: document.querySelector('.weather__max-temp'),
  weatherMinTemp: document.querySelector('.weather__min-tnemp'),
  forecastItems: document.querySelectorAll('.forecast-item'),
};

const tempValues = {
  city: '',
  labels: [],
  data: [],

  myChart: null,
};


function cityInputHandler(e) {
  tempValues.city = e.currentTarget.value.toLowerCase();
  console.log('City input:', tempValues.city)
}

function searchBtnClickHandler(e) {
  e.preventDefault();
  if (!tempValues.city) {
    Notify.failure('Please enter a city before searching.');
    return;
  }
  console.log(”Searching for:”, tempValues.city);
  fetchService(tempValues.city);
}
function fetchService(city) {
  promise.all([
  fetchWeatherData(city)])
    .then(data => {
      console.log(’Fetching weather data for:’, city);
      console.log(”Response data:”, data);
      refs.mainEl.className = ’’;

      if (tempValues.myChart) {
        console.log(’Destroying previous chart’);
        tempValues.myChart.destroy();
      }
      ’’
      if (data.error) {
        Notify.failure(data.error);  // If the API returns an error, notify the user
      } else {
      createsMarkup(data.weather, data.advisory, refs, tempValues);
      createsChart(tempValues.data);
    }
    })
    .catch(err => {
      Notify.failure(’Enter a valid city or crop type’);
    })
    .finally(() => {
      resetsValues();
    });
}
function resetsValues() {
  tempValues.city = ’’;
  tempValues.data = [];
  tempValues.labels = [];
  refs.cityInput.value = ’’;
}

weatherservice.js:

export async function fetchWeatherData(city) {
  try {
      const response = await fetch(‘http://127.0.0.1:5000/get_weather?city=${city}‘);
      if (!response.ok) throw new Error(’Failed to fetch weather data’);
      return await response.json();
  } catch (error) {
      console.error(’Error:’, error);
      Notiflix.Notify.failure(’Could not fetch weather data.’);
      throw error;
  }
}

export async function fetchWeatherAdvisory(cropType, hour) {
  try {
      const response = await fetch(‘/get_weather_advisory?crop_type=${cropType}&hour=${hour}‘);
      if (!response.ok) throw new Error(’Failed to fetch advisory’);
      return await response.json();
  } catch (error) {
      console.error(’Error:’, error);
      Notiflix.Notify.failure(’Could not fetch weather advisory.’);
      throw error;
  }
}

renderwebsite.js:

export function createsMarkup(weatherData,advisory ,obj, refs, tempValues) {
  refs.currentTemp.innerHTML = obj.current.temp_c;
  refs.feelslikeTemp.innerHTML = obj.current.feelslike_c;
  refs.cloudsValue.innerHTML = obj.current.cloud;
  refs.humidityValue.innerHTML = obj.current.humidity;
  refs.uvIdx.innerHTML = obj.current.uv;
  refs.weatherTitle.innerHTML = obj.current.condition.text;
  refs.weatherCity.innerHTML = obj.location.name;
  refs.weatherCountry.innerHTML = obj.location.country;
  refs.weatherDate.innerHTML = obj.location.localtime;
  refs.weatherWindDir.innerHTML = obj.current.wind_dir;
  refs.weatherSpeed.innerHTML = obj.current.wind_kph;
  refs.weatherPressure.innerHTML = obj.current.pressure_mb;
  refs.weatherRain.innerHTML =
    obj.forecast.forecastday[0].day.daily_chance_of_rain;
  refs.weatherSnow.innerHTML =
    obj.forecast.forecastday[0].day.daily_chance_of_snow;
  refs.weatherMaxTemp.innerHTML = obj.forecast.forecastday[0].day.maxtemp_c;
  refs.weatherMinTemp.innerHTML = obj.forecast.forecastday[0].day.mintemp_c;

  function createsStartForecastTime() {
    return +obj.location.localtime.split(’ ’)[1].split(’:’)[0] + 1;
  }

  createsForecastMarkup(createsForecastObj());

  function createsForecastObj() {
    const resultArray = obj.forecast.forecastday[0].hour.filter(
      (el) =>
        +el.time.split(’ ’)[1].split(’:’)[0] > +createsStartForecastTime()
    );

    return resultArray.length < 5
```

```
      ? obj.forecast.forecastday[0].hour.slice(-5)
      : resultArray.slice(0, 5);
  }

  function createsForecastMarkup(obj) {
    [...refs.forecastItems].forEach((el, idx) => {
      el.querySelector('.forecast__time').innerHTML =
        obj[idx].time.split(' ')[1];
      el.querySelector('.forecast__temperature--value').innerHTML =
        obj[idx].temp_c;
      el.querySelector('.forecast__wind-value').innerHTML = obj[idx].wind_kph;

      tempValues.labels.push(obj[idx].time.split(' ')[1]);
      tempValues.data.push(obj[idx].temp_c);
    });
  }

  const weatherConditions = [
    'sunny',
    'cloudy',
    'overcast',
    'mist',
    'rain',
    'snow',
    'sleet',
    'drizzle',
    'thundery',
    'thunder',
    'blizzard',
    'fog',
    'ice',
    'shower',
    'showers',
    'clear',
  ];

  let weatherConditionWord = obj.current.condition.text
    .toLowerCase()
    .split(' ')
    .find(el => weatherConditions.includes(el));

  refs.mainEl.classList.add(weatherConditionWord);
}

chartservice.js:

import Chart from 'chart.js/auto';

export function createsChart(tempValues) {
  if (tempValues.myChart) {
```

```
  tempValues.myChart.destroy(); // Destroy existing chart if it exists
}
  let ctx = document.getElementById('chart').getContext('2d');

  let gradient = ctx.createLinearGradient(0, -10, 0, 100);
  gradient.addColorStop(0, 'rgba(250, 0, 0, 1)');
  gradient.addColorStop(1, 'rgba(136, 255, 0, 1)');

  // Ensure labels and data are populated correctly
  if (tempValues.labels.length === 0 || tempValues.data.length === 0) {
    console.warn('No data available for chart');
    return; // Exit if there's no data to display
  }

  tempValues.myChart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: tempValues.labels,
      datasets: [
        {
          label: 'Celcius Degrees',
          data: tempValues.data,
          borderColor: gradient,
          borderWidth: 2,
          tension: 0.4,
          pointRadius: 2,
          yAxisID: 'y1',
        },
      ],
    },
    options: {
      plugins: {
        legend: {
          display: false,
        },
      },
      scales: {
        x: {
          display: false,
          grid: {
            drawOnChartArea: false,
          },
          gridLines: {
            display: false,
          },
        },
        y1: {
          type: 'linear',
          display: false,
          position: 'left',
          title: {
            display: false,
            text: 'Capital Partners',
          },
          ticks: {
            color: 'transparent',
          },
          grid: {
            drawOnChartArea: false,
          },
          gridLines: {
            display: false,
          },
        },
      },
      animation: {
        duration: 750,
      },
    },
  });
}
```

A.3.5 JSON Code

```
  {
”name”: ”weather-advisory”,
”version”: ”1.0.0”,
”main”: ”script.js”,
”type”: ”module”,
”dependencies”: {
  ”abab”: ”ˆ2.0.6”,
  ”abortcontroller-polyfill”: ”ˆ1.7.5”,
  ”acorn”: ”ˆ7.4.1”,
  ”acorn-globals”: ”ˆ4.3.4”,
  ”acorn-walk”: ”ˆ6.2.0”,
  ”ajv”: ”ˆ6.12.6”,
  ”alphanum-sort”: ”ˆ1.0.2”,
  ”ansi-regex”: ”ˆ2.1.1”,
  ”ansi-styles”: ”ˆ4.3.0”,
  ”ansi-to-html”: ”ˆ0.6.15”,
  ”anymatch”: ”ˆ2.0.0”,
  ”argparse”: ”ˆ2.0.1”,
  ”arr-diff”: ”ˆ4.0.0”,
  ”arr-flatten”: ”ˆ1.1.0”,
  ”arr-union”: ”ˆ3.1.0”,
  ”array-buffer-byte-length”: ”ˆ1.0.1”,
  ”array-equal”: ”ˆ1.0.2”,
  ”array-unique”: ”ˆ0.3.2”,
  ”array.prototype.reduce”: ”ˆ1.0.7”,
  ”arraybuffer.prototype.slice”: ”ˆ1.0.3”,
  ”asn1”: ”ˆ0.2.6”,
  ”asn1.js”: ”ˆ4.10.1”,
  ”assert”: ”ˆ1.5.1”,
  ”assert-plus”: ”ˆ1.0.0”,
  ”assign-symbols”: ”ˆ1.0.0”,
  ”async-each”: ”ˆ1.0.6”,
  ”async-limiter”: ”ˆ1.0.1”,
  ”asynckit”: ”ˆ0.4.0”,
  ”atob”: ”ˆ2.1.2”,
  ”available-typed-arrays”: ”ˆ1.0.7”,
  ”aws-sign2”: ”ˆ0.7.0”,
  ”aws4”: ”ˆ1.13.2”,
  ”axios”: ”ˆ1.7.7”,
  ”babel-plugin-polyfill-corejs2”: ”ˆ0.4.11”,
  ”babel-plugin-polyfill-corejs3”: ”ˆ0.10.6”,
  ”babel-plugin-polyfill-regenerator”: ”ˆ0.6.2”,
  ”babel-runtime”: ”ˆ6.26.0”,
  ”babel-types”: ”ˆ6.26.0”,
  ”babylon-walk”: ”ˆ1.0.2”,
  ”balanced-match”: ”ˆ1.0.2”,
  ”base”: ”ˆ0.11.2”,
  ”base-x”: ”ˆ3.0.10”,
  ”base64-js”: ”ˆ1.5.1”,
  ”bcrypt-pbkdf”: ”ˆ1.0.2”,
  ”binary-extensions”: ”ˆ1.13.1”,
  ”bindings”: ”ˆ1.5.0”,
  ”bn.js”: ”ˆ5.2.1”,
  ”boolbase”: ”ˆ1.0.0”,
  ”brace-expansion”: ”ˆ1.1.11”,
  ”braces”: ”ˆ3.0.3”,
  ”brfs”: ”ˆ1.6.1”,
  ”brorand”: ”ˆ1.1.0”,
  ”browser-process-hrtime”: ”ˆ1.0.0”,
  ”browserify-aes”: ”ˆ1.2.0”,
  ”browserify-cipher”: ”ˆ1.0.1”,
  ”browserify-des”: ”ˆ1.0.2”,
  ”browserify-rsa”: ”ˆ4.1.1”,
  ”browserify-sign”: ”ˆ4.2.3”,
  ”browserify-zlib”: ”ˆ0.2.0”,
  ”browserslist”: ”ˆ4.24.2”,
  ”buffer”:”ˆ6.0.3”,
  ”buffer-equal”: ”ˆ0.0.1”,
  ”buffer-from”: ”ˆ1.1.2”,
  ”buffer-xor”: ”ˆ1.0.3”,
  ”builtin-status-codes”: ”ˆ3.0.0”,
  ”cache-base”: ”ˆ1.0.1”,
  ”call-bind”: ”ˆ1.0.7”,
  ”call-me-maybe”: ”ˆ1.0.2”,
  ”caller-callsite”: ”ˆ2.0.0”,
  ”caller-path”: ”ˆ2.0.0”,
  ”callsites”: ”ˆ3.1.0”,
  ”caniuse-api”: ”ˆ3.0.0”,
  ”caniuse-lite”: ”ˆ1.0.30001677”,
  ”caseless”: ”ˆ0.12.0”,
  ”chalk”: ”ˆ4.1.2”,
  ”chart.js”: ”ˆ4.4.6”,
  ”chokidar”: ”ˆ4.0.1”,
  ”chrome-trace-event”: ”ˆ1.0.4”,
  ”cipher-base”: ”ˆ1.0.4”,
  ”class-utils”: ”ˆ0.3.6”,
  ”cli-cursor”: ”ˆ2.1.0”,
  ”cli-spinners”: ”ˆ1.3.1”,
  ”clone”: ”ˆ2.1.2”,
  ”coa”: ”ˆ2.0.2”,
  ”collection-visit”: ”ˆ1.0.0”,
  ”color”: ”ˆ3.2.1”,
  ”color-convert”: ”ˆ2.0.1”,
  ”color-name”: ”ˆ1.1.4”,
  ”color-string”: ”ˆ1.9.1”,
  ”colord”: ”ˆ2.9.3”,
  ”combined-stream”: ”ˆ1.0.8”,
  ”command-exists”: ”ˆ1.2.9”,
  ”commander”: ”^7.2.0”,
  ”component-emitter”: ”^1.3.1”,
  ”concat-map”: ”^0.0.1”,
  ”concat-stream”: ”^1.6.2”,
  ”console-browserify”: ”^1.2.0”,
  ”constants-browserify”: ”^1.0.0”,
  ”convert-source-map”: ”^2.0.0”,
  ”copy-descriptor”: ”^0.1.1”,
  ”core-js”: ”^2.6.12”,
  ”core-js-compat”: ”^3.39.0”,
  ”core-util-is”: ”^1.0.3”,
  ”cosmiconfig”: ”^9.0.0”,
  ”create-ecdh”: ”^4.0.4”,
  ”create-hash”: ”^1.2.0”,
  ”create-hmac”: ”^1.1.7”,
  ”cross-spawn”: ”^6.0.5”,
  ”crypto-browserify”: ”^3.12.1”,
  ”css-color-names”: ”^0.0.4”,
  ”css-declaration-sorter”: ”^7.2.0”,
  ”css-modules-loader-core”: ”^1.1.0”,
  ”css-select”: ”^5.1.0”,
  ”css-select-base-adapter”: ”^0.1.1”,
  ”css-selector-tokenizer”: ”^0.7.3”,
  ”css-tree”: ”^2.3.1”,
  ”css-what”: ”^6.1.0”,
  ”cssesc”: ”^3.0.0”,
  ”cssnano”: ”^7.0.6”,
  ”cssnano-preset-default”: ”^7.0.6”,
  ”cssnano-util-get-arguments”: ”^4.0.0”,
  ”cssnano-util-get-match”: ”^4.0.0”,
  ”cssnano-util-raw-cache”: ”^4.0.1”,
  ”cssnano-util-same-parent”: ”^4.0.1”,
  ”cssnano-utils”: ”^5.0.0”,
  ”csso”: ”^5.0.5”,
  ”cssom”: ”^0.3.8”,
  ”cssstyle”: ”^1.4.0”,
  ”dashdash”: ”^1.14.1”,
  ”data-urls”: ”^1.1.0”,
  ”data-view-buffer”: ”^1.0.1”,
  ”data-view-byte-length”: ”^1.0.1”,
  ”data-view-byte-offset”: ”^1.0.0”,
  ”deasync”: ”^0.1.30”,
  ”debug”: ”^4.3.7”,
  ”decode-uri-component”: ”^0.2.2”,
  ”deep-is”: ”^0.1.4”,
  ”defaults”: ”^1.0.4”,
  ”define-data-property”: ”^1.1.4”,
  ”define-properties”: ”^1.2.1”,
  ”define-property”: ”^2.0.2”,
  ”delayed-stream”: ”^1.0.0”,
  "depd": "^2.0.0",
  "des.js": "^1.1.0",
  "destroy": "^1.2.0",
  "detect-libc": "^1.0.3",
  "diffie-hellman": "^5.0.3",
  "dom-serializer": "^1.4.1",
  "domain-browser": "^1.2.0",
  "domelementtype": "^2.3.0",
  "domexception": "^1.0.1",
  "domhandler": "^4.3.1",
  "domutils": "^2.8.0",
  "dot-prop": "^5.3.0",
  "dotenv": "^7.0.0",
  "dotenv-expand": "^5.1.0",
  "duplexer2": "^0.1.4",
  "eastasianwidth": "^0.2.0",
  "ecc-jsbn": "^0.1.2",
  "ee-first": "^1.1.1",
  "electron-to-chromium": "^1.5.50",
  "elliptic": "^6.6.0",
  "emoji-regex": "^9.2.2",
  "encodeurl": "^2.0.0",
  "entities": "^2.2.0",
  "env-paths": "^2.2.1",
  "envinfo": "^7.14.0",
  "error-ex": "^1.3.2",
  "es-abstract": "^1.23.3",
  "es-array-method-boxes-properly": "^1.0.0",
  "es-define-property": "^1.0.0",
  "es-errors": "^1.3.0",
  "es-object-atoms": "^1.0.0",
  "es-set-tostringtag": "^2.0.3",
  "es-to-primitive": "^1.2.1",
  "escalade": "^3.2.0",
  "escape-html": "^1.0.3",
  "escape-string-regexp": "^1.0.5",
  "escodegen": "^1.9.1",
  "esprima": "^4.0.1",
  "estraverse": "^4.3.0",
  "esutils": "^2.0.3",
  "etag": "^1.8.1",
  "events": "^3.3.0",
  "evp_bytestokey": "^1.0.3",
  "expand-brackets": "^2.1.4",
  "extend": "^3.0.2",
  "extend-shallow": "^3.0.2",
  "extglob": "^2.0.4",
  "extsprintf": "^1.3.0",
  "falafel": "^2.2.5",
  "fast-deep-equal": "^3.1.3",
  "fast-glob": "^2.2.7",
  "fast-json-stable-stringify": "^2.1.0",
  "fast-levenshtein": "^2.0.6",
  "fastparse": "^1.1.2",
  "fastq": "^1.17.1",
  "fclone": "^1.0.11",
  "file-uri-to-path": "^1.0.0",
  "filesize": "^3.6.1",
  "fill-range": "^7.1.1",
  "follow-redirects": "^1.15.9",
  "for-each": "^0.3.3",
  "for-in": "^1.0.2",
  "foreground-child": "^3.3.0",
  "forever-agent": "^0.6.1",
  "form-data": "^4.0.1",
  "fragment-cache": "^0.2.1",
  "fresh": "^0.5.2",
  "fs.realpath": "^1.0.0",
  "function-bind": "^1.1.2",
  "function.prototype.name": "^1.1.6",
  "functions-have-names": "^1.2.3",
  "gensync": "^1.0.0-beta.2",
  "get-intrinsic": "^1.2.4",
  "get-port": "^4.2.0",
  "get-symbol-description": "^1.0.2",
  "get-value": "^2.0.6",
  "getpass": "^0.1.7",
  "glob": "^7.2.3",
  "glob-parent": "^3.1.0",
  "glob-to-regexp": "^0.3.0",
  "globals": "^13.24.0",
  "globalthis": "^1.0.4",
  "gopd": "^1.0.1",
  "graceful-fs": "^4.2.11",
  "grapheme-breaker": "^0.3.2",
  "har-schema": "^2.0.0",
  "har-validator": "^5.1.5",
  "has": "^1.0.4",
  "has-ansi": "^2.0.0",
  "has-bigints": "^1.0.2",
  "has-flag": "^4.0.0",
  "has-property-descriptors": "^1.0.2",
  "has-proto": "^1.0.3",
  "has-symbols": "^1.0.3",
  "has-tostringtag": "^1.0.2",
  "has-value": "^1.0.0",
  "has-values": "^1.0.0",
  "hash-base": "^3.0.4",
  "hash.js": "^1.1.7",
  "hasown": "^2.0.2",
  ”hex-color-regex”: ”^1.1.0”,
  ”hmac-drbg”: ”^1.0.1”,
  ”hsl-regex”: ”^1.0.0”,
  ”hsla-regex”: ”^1.0.0”,
  ”html-encoding-sniffer”: ”^1.0.2”,
  ”html-tags”: ”^1.2.0”,
  ”htmlnano”: ”^2.1.1”,
  ”htmlparser2”: ”^7.2.0”,
  ”http-errors”: ”^2.0.0”,
  ”http-signature”: ”^1.2.0”,
  ”https-browserify”: ”^1.0.0”,
  ”iconv-lite”: ”^0.4.24”,
  ”icss-replace-symbols”: ”^1.1.0”,
  ”ieee754”: ”^1.2.1”,
  ”immutable”: ”^4.3.7”,
  ”import-fresh”: ”^3.3.0”,
  ”indexes-of”: ”^1.0.1”,
  ”inflight”: ”^1.0.6”,
  ”is-binary-path”: ”^1.0.1”,
  ”is-boolean-object”: ”^1.1.2”,
  ”is-buffer”: ”^1.1.6”,
  ”is-callable”: ”^1.2.7”,
  ”is-color-stop”: ”^1.1.0”,
  ”is-core-module”: ”^2.15.1”,
  ”is-data-descriptor”: ”^1.0.1”,
  ”is-data-view”: ”^1.0.1”,
  ”is-date-object”: ”^1.0.5”,
  ”is-descriptor”: ”^1.0.3”,
  ”is-directory”: ”^0.3.1”,
  ”is-extendable”: ”^1.0.1”,
  ”is-extglob”: ”^2.1.1”,
  ”is-fullwidth-code-point”: ”^3.0.0”,
  ”is-glob”: ”^4.0.3”,
  ”is-html”: ”^1.1.0”,
  ”is-json”: ”^2.0.1”,
  ”is-negative-zero”: ”^2.0.3”,
  ”is-number”: ”^7.0.0”,
  ”is-number-object”: ”^1.0.7”,
  ”is-obj”: ”^2.0.0”,
  ”is-plain-object”: ”^2.0.4”,
  ”is-regex”: ”^1.1.4”,
  ”is-resolvable”: ”^1.1.0”,
  ”is-shared-array-buffer”: ”^1.0.3”,
  ”is-string”: ”^1.0.7”,
  ”is-symbol”: ”^1.0.4”,
  ”is-typed-array”: ”^1.1.13”,
  ”is-typedarray”: ”^1.0.0”,
  ”is-url”: ”^1.2.4”,
  ”is-weakref”: ”^1.0.2”,
  ”is-windows”: ”^1.0.2”,
  "is-wsl": "^1.1.0",
  "isarray": "^1.0.0",
  "isexe": "^2.0.0",
  "isobject": "^3.0.1",
  "isstream": "^0.1.2",
  "jackspeak": "^3.4.3",
  "js-tokens": "^4.0.0",
  "js-yaml": "^4.1.0",
  "jsbn": "^0.1.1",
  "jsdom": "^14.1.0",
  "jsesc": "^3.0.2",
  "json-parse-better-errors": "^1.0.2",
  "json-parse-even-better-errors": "^2.3.1",
  "json-schema": "^0.4.0",
  "json-schema-traverse": "^0.4.1",
  "json-stringify-safe": "^5.0.1",
  "json5": "^2.2.3",
  "jsprim": "^1.4.2",
  "kind-of": "^6.0.3",
  "levn": "^0.3.0",
  "lightningcss": "^1.27.0",
  "lightningcss-win32-x64-msvc": "^1.27.0",
  "lilconfig": "^3.1.2",
  "lines-and-columns": "^1.2.4",
  "lmdb": "^2.8.5",
  "lodash": "^4.17.21",
  "lodash.clone": "^4.5.0",
  "lodash.debounce": "^4.0.8",
  "lodash.memoize": "^4.1.2",
  "lodash.sortby": "^4.7.0",
  "lodash.uniq": "^4.5.0",
  "log-symbols": "^2.2.0",
  "lru-cache": "^5.1.1",
  "magic-string": "^0.22.5",
  "map-cache": "^0.2.2",
  "map-visit": "^1.0.0",
  "md5.js": "^1.3.5",
  "mdn-data": "^2.0.30",
  "merge-source-map": "^1.0.4",
  "merge2": "^1.4.1",
  "micromatch": "^4.0.8",
  "miller-rabin": "^4.0.1",
  "mime": "^1.6.0",
  "mime-db": "^1.52.0",
  "mime-types": "^2.1.35",
  "mimic-fn": "^1.2.0",
  "minimalistic-assert": "^1.0.1",
  "minimalistic-crypto-utils": "^1.0.1",
  "minimatch": "^3.1.2",
  "minimist": "^1.2.8",
  "minipass": "^7.1.2",
  "mixin-deep": "^1.3.2",
  "mkdirp": "^0.5.6",
  "modern-normalize": "^1.1.0",
  "ms": "^2.1.3",
  "msgpackr": "^1.11.2",
  "msgpackr-extract": "^3.0.3",
  "nanoid": "^3.3.7",
  "nanomatch": "^1.2.13",
  "nice-try": "^1.0.5",
  "node-addon-api": "^7.1.1",
  "node-forge": "^0.10.0",
  "node-gyp-build-optional-packages": "^5.1.1",
  "node-libs-browser": "^2.2.1",
  "node-releases": "^2.0.18",
  "normalize-path": "^3.0.0",
  "normalize-url": "^3.3.0",
  "notiflix": "^3.2.7",
  "nth-check": "^2.1.1",
  "nullthrows": "^1.1.1",
  "nwsapi": "^2.2.13",
  "oauth-sign": "^0.9.0",
  "object-copy": "^0.1.0",
  "object-inspect": "^1.13.2",
  "object-keys": "^1.1.1",
  "object-visit": "^1.0.1",
  "object.assign": "^4.1.5",
  "object.getownpropertydescriptors": "^2.1.8",
  "object.pick": "^1.3.0",
  "object.values": "^1.2.0",
  "on-finished": "^2.4.1",
  "once": "^1.4.0",
  "onetime": "^2.0.1",
  "opn": "^5.5.0",
  "optionator": "^0.8.3",
  "ora": "^2.1.0",
  "ordered-binary": "^1.5.3",
  "os-browserify": "^0.3.0",
  "package-json-from-dist": "^1.0.1",
  "pako": "^1.0.11",
  "parcel": "^2.12.0",
  "parcel-bundler": "^1.12.5",
  "parcel-reporter-clean-dist": "^1.0.4",
  "parent-module": "^1.0.1",
  "parse-asn1": "^5.1.7",
  "parse-json": "^5.2.0",
  "parse5": "^5.1.0",
  "parseurl": "^1.3.3",
  "pascalcase": "^0.1.1",
  "path-browserify": "^0.0.1",
  "path-dirname": "^1.0.2",
  "path-is-absolute": "^1.0.1",
  "path-key": "^2.0.1",
  "path-parse": "^1.0.7",
  "path-scurry": "^1.11.1",
  "pbkdf2": "^3.1.2",
  "performance-now": "^2.1.0",
  "physical-cpu-count": "^2.0.0",
  "picocolors": "^1.1.1",
  "picomatch": "^2.3.1",
  "pn": "^1.1.0",
  "posix-character-classes": "^0.1.1",
  "possible-typed-array-names": "^1.0.0",
  "postcss": "^8.4.47",
  "postcss-calc": "^10.0.2",
  "postcss-colormin": "^7.0.2",
  "postcss-convert-values": "^7.0.4",
  "postcss-discard-comments": "^7.0.3",
  "postcss-discard-duplicates": "^7.0.1",
  "postcss-discard-empty": "^7.0.0",
  "postcss-discard-overridden": "^7.0.0",
  "postcss-merge-longhand": "^7.0.4",
  "postcss-merge-rules": "^7.0.4",
  "postcss-minify-font-values": "^7.0.0",
  "postcss-minify-gradients": "^7.0.0",
  "postcss-minify-params": "^7.0.2",
  "postcss-minify-selectors": "^7.0.4",
  "postcss-modules-extract-imports": "^3.1.0",
  "postcss-modules-local-by-default": "^4.0.5",
  "postcss-modules-scope": "^3.2.0",
  "postcss-modules-values": "^1.3.0",
  "postcss-normalize-charset": "^7.0.0",
  "postcss-normalize-display-values": "^7.0.0",
  "postcss-normalize-positions": "^7.0.0",
  "postcss-normalize-repeat-style": "^7.0.0",
  "postcss-normalize-string": "^7.0.0",
  "postcss-normalize-timing-functions": "^7.0.0",
  "postcss-normalize-unicode": "^7.0.2",
  "postcss-normalize-url": "^7.0.0",
  "postcss-normalize-whitespace": "^7.0.0",
  "postcss-ordered-values": "^7.0.1",
  "postcss-reduce-initial": "^7.0.2",
  "postcss-reduce-transforms": "^7.0.0",
  "postcss-selector-parser": "^6.1.2",
  "postcss-svgo": "^7.0.1",
  "postcss-unique-selectors": "^7.0.3",
  "postcss-value-parser": "^4.2.0",
  "posthtml": "^0.16.6",
  "posthtml-expressions": "^1.11.4",
  "posthtml-include": "^1.7.4",
  ”posthtml-match-helper”: ”ˆ1.0.4”,
  ”posthtml-parser”: ”ˆ0.10.2”,
  ”posthtml-render”: ”ˆ3.0.0”,
  ”prelude-ls”: ”ˆ1.1.2”,
  ”process”: ”ˆ0.11.10”,
  ”process-nextick-args”: ”ˆ2.0.1”,
  ”proxy-from-env”: ”ˆ1.1.0”,
  ”psl”: ”ˆ1.9.0”,
  ”public-encrypt”: ”ˆ4.0.3”,
  ”punycode”: ”ˆ1.4.1”,
  ”purgecss”: ”ˆ6.0.0”,
  ”q”: ”ˆ1.5.1”,
  ”qs”: ”ˆ6.13.0”,
  ”querystring-es3”: ”ˆ0.2.1”,
  ”queue-microtask”: ”ˆ1.2.3”,
  ”quote-stream”: ”ˆ1.0.2”,
  ”randombytes”: ”ˆ2.1.0”,
  ”randomfill”: ”ˆ1.0.4”,
  ”range-parser”: ”ˆ1.2.1”,
  ”react-error-overlay”: ”ˆ6.0.9”,
  ”react-refresh”: ”ˆ0.9.0”,
  ”readable-stream”: ”ˆ2.3.8”,
  ”readdirp”: ”ˆ4.0.2”,
  ”regenerate”: ”ˆ1.4.2”,
  ”regenerate-unicode-properties”: ”ˆ10.2.0”,
  ”regenerator-runtime”: ”ˆ0.13.11”,
  ”regenerator-transform”: ”ˆ0.15.2”,
  ”regex-not”: ”ˆ1.0.2”,
  ”regexp.prototype.flags”: ”ˆ1.5.3”,
  ”regexpu-core”: ”ˆ6.1.1”,
  ”regjsgen”: ”ˆ0.8.0”,
  ”regjsparser”: ”ˆ0.11.2”,
  ”relateurl”: ”ˆ0.2.7”,
  ”remove-trailing-separator”: ”ˆ1.1.0”,
  ”repeat-element”: ”ˆ1.1.4”,
  ”repeat-string”: ”ˆ1.6.1”,
  ”request”: ”ˆ2.88.2”,
  ”request-promise-core”: ”ˆ1.1.4”,
  ”request-promise-native”: ”ˆ1.0.9”,
  ”resolve”: ”ˆ1.22.8”,
  ”resolve-from”: ”ˆ4.0.0”,
  ”resolve-url”: ”ˆ0.2.1”,
  ”restore-cursor”: ”ˆ2.0.0”,
  ”ret”: ”ˆ0.1.15”,
  ”reusify”: ”ˆ1.0.4”,
  ”rgb-regex”: ”ˆ1.0.1”,
  ”rgba-regex”: ”ˆ1.0.0”,
  ”rimraf”: ”ˆ2.7.1”,
  ”ripemd160”: ”ˆ2.0.2”,
  ”run-parallel”: ”ˆ1.2.0”,
  "safe-array-concat": "^1.1.2",
  "safe-buffer": "^5.2.1",
  "safe-regex": "^1.1.0",
  "safe-regex-test": "^1.0.3",
  "safer-buffer": "^2.1.2",
  "sass": "^1.80.6",
  "sax": "^1.2.4",
  "saxes": "^3.1.11",
  "semver": "^7.6.3",
  "send": "^0.19.0",
  "serialize-to-js": "^3.1.2",
  "serve-static": "^1.16.2",
  "set-function-length": "^1.2.2",
  "set-function-name": "^2.0.2",
  "set-value": "^2.0.1",
  "setimmediate": "^1.0.5",
  "setprototypeof": "^1.2.0",
  "sha.js": "^2.4.11",
  "shallow-copy": "^0.0.1",
  "shebang-command": "^1.2.0",
  "shebang-regex": "^1.0.0",
  "side-channel": "^1.0.6",
  "signal-exit": "^3.0.7",
  "simple-swizzle": "^0.2.2",
  "snapdragon": "^0.8.2",
  "snapdragon-node": "^2.1.1",
  "snapdragon-util": "^3.0.1",
  "source-map": "^0.6.1",
  "source-map-js": "^1.2.1",
  "source-map-resolve": "^0.5.3",
  "source-map-support": "^0.5.21",
  "source-map-url": "^0.4.1",
  "split-string": "^3.1.0",
  "sprintf-js": "^1.0.3",
  "srcset": "^5.0.1",
  "sshpk": "^1.18.0",
  "stable": "^0.1.8",
  "static-eval": "^2.1.1",
  "static-extend": "^0.1.2",
  "static-module": "^2.2.5",
  "statuses": "^2.0.1",
  "stealthy-require": "^1.1.1",
  "stream-browserify": "^2.0.2",
  "stream-http": "^2.8.3",
  "string_decoder": "^1.3.0",
  "string-width": "^5.1.2",
  "string-width-cjs": "^4.2.3",
  "string.prototype.trim": "^1.2.9",
  "string.prototype.trimend": "^1.0.8",
  "string.prototype.trimstart": "^1.0.8",
  "strip-ansi": "^3.0.1",
  "strip-ansi-cjs": "^6.0.1",
  "stylehacks": "^7.0.4",
  "supports-color": "^7.2.0",
  "supports-preserve-symlinks-flag": "^1.0.0",
  "svgo": "^3.3.2",
  "symbol-tree": "^3.2.4",
  "term-size": "^2.2.1",
  "terser": "^5.36.0",
  "through2": "^2.0.5",
  "timers-browserify": "^2.0.12",
  "timsort": "^0.3.0",
  "tiny-inflate": "^1.0.3",
  "to-arraybuffer": "^1.0.1",
  "to-fast-properties": "^1.0.3",
  "to-object-path": "^0.3.0",
  "to-regex": "^3.0.2",
  "to-regex-range": "^5.0.1",
  "toidentifier": "^1.0.1",
  "tough-cookie": "^2.5.0",
  "tr46": "^1.0.1",
  "tslib": "^2.8.1",
  "tty-browserify": "^0.0.0",
  "tunnel-agent": "^0.6.0",
  "tweetnacl": "^0.14.5",
  "type-check": "^0.3.2",
  "type-fest": "^0.20.2",
  "typed-array-buffer": "^1.0.2",
  "typed-array-byte-length": "^1.0.1",
  "typed-array-byte-offset": "^1.0.2",
  "typed-array-length": "^1.0.6",
  "typedarray": "^0.0.6",
  "unbox-primitive": "^1.0.2",
  "uncss": "^0.17.3",
  "unicode-canonical-property-names-ecmascript": "^2.0.1",
  "unicode-match-property-ecmascript": "^2.0.0",
  "unicode-match-property-value-ecmascript": "^2.2.0",
  "unicode-property-aliases-ecmascript": "^2.1.0",
  "unicode-trie": "^0.3.1",
  "union-value": "^1.0.1",
  "uniq": "^1.0.1",
  "uniqs": "^2.0.0",
  "unquote": "^1.1.1",
  "unset-value": "^1.0.0",
  "upath": "^1.2.0",
  "update-browserslist-db": "^1.1.1",
  "uri-js": "^4.4.1",
  "urix": "^0.1.0",
  "url": "^0.11.4",
  "use": "^3.1.1",
  "util": "^0.11.1",
  "util-deprecate": "^1.0.2",
  "util.promisify": "^1.0.1",
  "utility-types": "^3.11.0",
  "uuid": "^3.4.0",
  "v8-compile-cache": "^2.4.0",
  "vendors": "^1.0.4",
  "verror": "^1.10.0",
  "vlq": "^0.2.3",
  "vm-browserify": "^1.1.2",
  "w3c-hr-time": "^1.0.2",
  "w3c-xmlserializer": "^1.1.2",
  "wcwidth": "^1.0.1",
  "weak-lru-cache": "^1.2.2",
  "webidl-conversions": "^4.0.2",
  "whatwg-encoding": "^1.0.5",
  "whatwg-mimetype": "^2.3.0",
  "whatwg-url": "^7.1.0",
  "which": "^1.3.1",
  "which-boxed-primitive": "^1.0.2",
  "which-typed-array": "^1.1.15",
  "word-wrap": "^1.2.5",
  "wrap-ansi": "^8.1.0",
  "wrap-ansi-cjs": "^7.0.0",
  "wrappy": "^1.0.2",
  "ws": "^5.2.4",
  "xml-name-validator": "^3.0.0",
  "xmlchars": "^2.2.0",
  "xtend": "^4.0.2",
  "yallist": "^3.1.1"
},
"devDependencies": {
  "parcel-bundler": "^latest-version"
},
"scripts": {
  "build": "parcel build templates/*.html",
  "dev": "parcel templates/*.html"
},
"keywords": [],
"author": "",
"license": "ISC",
"description": ""
}
```

References

[1] C. Zoremsanga and J. Hussain, ”Particle Swarm optimized Deep Learning Models for Rainfall Prediction: A Case Study in Aizawl, Mizoram,” in IEEE Access, vol. 12, pp. 57172-57184, 2024, doi: 10.1109/ACCESS.2024.3390781.

[2] E. Rocha Rodrigues, I. oliveira, R. Cunha and M. Netto, ”DeepDown-scale: A Deep Learning Strategy for High-Resolution Weather Forecast,” 2018 IEEE 14th International Conference on e-Science (e-Science), Amsterdam, Netherlands, 2018, pp. 415-422, doi:10.1109/eScience.2018.00130.

[3] H. Kim, S. Park, H. -J. Park, H. -G. Son and S. Kim, ”Solar Radiation Forecasting Based on the Hybrid CNN-CatBoost Model,” in IEEE Access, vol. 11, pp. 13492-13500, 2023, doi: 10.1109/ACCESS.2023.3243252.

[4] K. Chattrairat, W. Wongseree and A. Leelasantitham, ”Comparisons of Machine Learning Methods of Statistical Downscaling Method: Case Studies of Daily Climate Anomalies in Thailand,” in Journal of Web Engineering, vol. 20, no. 5, pp. 1459-1486, July 2021, doi: 10.13052/jwe1540-9589.2057.

[5] M. M. Hassan et al., ”Machine Learning-Based Rainfall Prediction : Unveiling Insights and Forecasting for Improved Preparedness,” in IEEE Access, vol. 11, pp. 132196-132222, 2023, doi: 10.1109/ACCESS.2023.3333876.

[6] Q. -D. -E. -J. Ren, N. Li and W. Zhang, ”Research on Sand-Dust Storm Forecasting Based on Deep Neural Network With Stacking Ensemble Learning,” in IEEE Access, vol. 10, pp. 111855-111863, 2022, doi: 10.1109/ACCESS.2022.3216309.

[7] R. Cai, S. Xie, B. Wang, R. Yang, D. Xu and Y. He, ”Wind Speed Forecasting Based on Extreme Gradient Boosting,” in IEEE Access, vol. 8, pp.175063-175069, 2020, doi: 10.1109/ACCESS.2020.3025967.

[8] X. Chu, W. Bai, Y. Sun, W. Li, C. Liu and H. Song, ”A Machine Learning-Based Method for Wind Fields Forecasting Utilizing GNSS Radio occultation Data,” in IEEE Access, vol. 10, pp. 30258-30273, 2022, doi:10.1109/ ACCESS. 2022. 3159231.

[9] Y. Essa, H. G. P. Hunt, M. Gijben and R. Ajoodha, ”Deep Learning Prediction of Thunderstorm Severity Using Remote Sensing Weather Data,” in IEEE Journal of Selected Topics in Applied Earth observations and Remote Sensing, vol. 15, pp. 4004-4013, 2022, doi: 10.1109/JSTARS.2022.3172785.

[10] Y. V. Wang et al., ”Nowcasting Heavy Rainfall With Convolutional Long Short-Term Memory Networks: A Pixelwise Modeling Approach,” in IEEE Journal of Selected Topics in Applied Earth observations and Remote Sensing, vol. 17, pp. 8424-8433, 2024, doi: 10.1109/JSTARS.2024.3383397.

[11] Y. V. Wang et al., ”Relative Importance of Radar Variables for Nowcasting Heavy Rainfall: A Machine Learning Approach,” in IEEE Transactions on Geoscience and Remote Sensing, vol. 61, pp. 1-14, 2023, Art no. 4100314, doi: 10.1109/ TGRS.2022.3231125.