INDUSTRIAL ORIENTED MINI PROJECT Report

on

SMART WATER SCARCITY ANALYSIS AND DISTRIBUTION SOLUTION

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

by

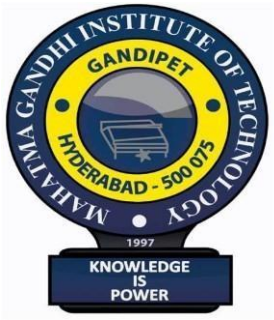
D Ashrith Reddy-22261A1216

Aruri Nikhil- 22261A1205

Under the guidance of

Mr. B. Lokesh

Assistant Professor , Department of IT



DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **SMART WATER SCARCITY ANALYSIS AND DISTRIBUTION SOLUTION** submitted by **D Ashrith Reddy (22261A1216), Aruri Nikhil (22261A1205)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mr. B. Lokesh** , and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:

Mr.B.Lokesh
Assistant Professor
Dept. of IT

IOMP Supervisor:

Dr. U. Chaitanya
Assistant Professor
Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hereby declare that the **Industrial Oriented Mini Project** entitled **SMART WATER SCARCITY ANALYSIS AND DISTRIBUTION SOLUTION** is an original and bonafide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mr.B. Lokesh, Assistant Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

D Ashrith Reddy-22261A1216

Aruri Nikhil-22261A1205

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our project guide **Mr. B. Lokesh , Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our Project Coordinator(s) **Dr. U. Chaitanya** , Assistant Professor ,Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

D Ashrith Reddy-22261A1216

Aruri Nikhil-22261A1205

ABSTRACT

Water scarcity is a growing concern worldwide, requiring efficient monitoring and distribution strategies for the growth and needs of common citizens as well as the entire society. The Smart Water Scarcity & Distribution System leverages Google Earth Engine (GEE), NASA GRACE, and GPM IMERG datasets to analyse groundwater levels and rainfall trends in real-time. This system provides precise water availability predictions, enabling authorities to plan optimal water distribution based on demand and scarcity levels.

The system utilizes remote sensing data to fetch historical and real-time water availability for any given location. It classifies regions into scarcity levels and predicts future water consumption using machine learning techniques. Real-time weather data from OpenWeatherMap API further enhances accuracy. Based on the collected data, the system generates visual trends and helps governments and municipalities distribute water efficiently to high-demand areas.

The system uses WorldPop and NASA SEDAC datasets as well to fetch the real time population and population density data to estimate the water consumption for each area based on which we can predict the water demand for every area based on the historical data.

TABLE OF CONTENTS

Chapter No	Title	Page No
	CERTIFICATE	**I**
	DECLARATION	**Ii**
	ACKNOWLEDGEMENT	**Iii**
	ABSTRACT	**iv**
	LIST OF FIGURES	**vii**
	LIST OF TABLES	**Viii**
1	**INTRODUCTION**	**1**
	1.1 MOTIVATION	1
	1.2 PROBLEM STATEMENT	1
	1.3 EXISTING SYSTEM	2
	1.3.1 LIMITATIONS	3
	1.4 PROPOSED SYSTEM	3
	1.4.1 ADVANTAGES	3
	1.5 OBJECTIVES	4
	1.6 HARDWARE AND SOFTWARE REQUIREMENTS	4
2	**LITERATURE SURVEY**	**6**
3	**ANALYSIS AND DESIGN**	**8**
	3.1 MODULES	8
	3.2 ARCHITECTURE	10
	3.3 UML DIAGRAMS	11
	3.3.1 USE CASE DIAGRAM	11
	3.3.2 CLASS DIAGRAM	13
	3.3.3 ACTIVITY DIAGRAM	15
	3.3.4 SEQUENCE DIAGRAM	18
	3.3.5 COMPONENT DIAGRAM	20
	3.3.6 DEPLOYMENT DIAGRAM	23

Chapter No	Title	Page No
	3.4 METHODOLOGY	24
4	**CODE AND IMPLEMENTATION**	**25**
	4.1 CODE	25
	4.2 IMPLEMENTATION	42
5	**TESTING**	**44**
	5.1 INTRODUCTION TO TESTING	44
	5.2 TEST CASES	45
6	**RESULTS**	**46**
7	**CONCLUSION AND FUTURE ENHANCEMENTS**	50
	7.1 CONCLUSION	**50**
	7.2 FUTURE ENHANCEMENTS	50
8	**REFERENCES**	52

LIST OF FIGURES

Fig No	Fig Name	Page No
Fig. 3.2.1	Architecture of Smart Water Analysis	10
Fig. 3.3.1.1	Use Case Diagram	12
Fig. 3.3.2.1	Class Diagram	13
Fig 3.3.3.1	Activity Diagram	16
Fig 3.3.4.1	Sequence Diagram	18
Fig 3.3.5.1	Component Diagram	20
Fig. 3.3.6.1	Deployment Diagram	22
Fig 6.1	Dropdown Selection	40
Fig. 6.2	Weather Data	40
Fig. 6.3	Color Legend	41
Fig. 6.4	Groundwater Scarcity Map	41
Fig 6.5	Estimated Water Demand Map	42
Fig. 6.6	Groundwater Level Forecast	43
Fig. 6.7	Water Demand Forecast	43

LIST OF TABLES

Table 2.1 Literature Survey of Research papers 7
Table 5.1 Test Cases of Smart Water Analysis 45

1. INTRODUCTION

1.1 MOTIVATION

Water scarcity and inefficient resource management are growing challenges, especially in regions dependent on groundwater for agriculture and daily use. Traditional water monitoring systems are often fragmented, reactive, and data-poor, leading to poor planning and over-extraction. Recognizing the urgency of this issue, our Smart Water Monitoring and Prediction project aims to provide a data-driven solution to forecast groundwater levels, assess scarcity, and estimate future water demand — enabling more sustainable and informed decision-making.

This project integrates machine learning, geospatial mapping, and real-time weather APIs to create an interactive platform that visualizes groundwater trends and predicts future conditions. By analyzing historic data along with climate indicators like rainfall and temperature, we can forecast both groundwater levels and district-wise water demand. The tool empowers governments, planners, and even citizens with clear, visual, and predictive information that is typically buried in spreadsheets and static reports.

Beyond analytics, the project is designed to raise awareness about water stress at a local level. By visualizing which districts are nearing critical scarcity and projecting demand growth, it encourages timely intervention and smarter allocation of water resources.

1.2 PROBLEM STATEMENT

Groundwater is a primary source of water for agriculture, industry, and domestic consumption in many parts of India, especially in states like Telangana and Andhra Pradesh. However, over-extraction and insufficient recharge are causing groundwater levels to decline at an alarming rate. Existing monitoring systems are limited, often manual, and lack real-time insight, making it difficult to anticipate water shortages or plan efficient usage. Without accessible, localized data, stakeholders remain unaware of when regions shift from safe to critical scarcity

Though various government bodies collect water, climate, and population data, these datasets are siloed, outdated, and rarely analysed together. There is little effort to integrate weather patterns, recharge potential, and population growth to forecast future water demand

or groundwater availability. As a result, water policies are often reactive instead of preventive, leading to inefficient distribution and delayed responses to droughts or overuse.

There is an urgent need for a smart, visual, and data-driven solution that can monitor current groundwater conditions, predict future risks, and guide both policy and public awareness. Without such tools, sustainable water management remains out of reach.

1.3 EXISTING SYSTEM

Government agencies like the Central Ground Water Board (CGWB) collect groundwater data through observation wells. While useful, the data is often limited to monthly updates and lacks granularity. These systems are not real-time and are difficult for the public to access. They also rarely integrate climate or demand data for forecasting.

In many regions, water level readings are recorded manually and stored in local registers. These records are rarely digitized or analysed, leading to loss of valuable trends over time. Decision-making remains reactive rather than predictive. Most local bodies lack the technical tools to visualize or interpret this data effectively.

Some dashboards and GIS maps show historical groundwater data district-wise. However, they mostly offer static views without machine learning-based forecasting or real-time updates. They don’t account for rainfall, temperature, or population growth. As a result, they fail to support proactive planning or early warning.

1.3.1 LIMITATIONS

- **Lack of Real-Time and Predictive Capability**: Most existing systems provide only historical or monthly water level data, lacking real-time monitoring or forecasting features. This prevents early identification of water stress and offers little time for intervention. Without predictive models, planning for future demand or scarcity remains guesswork.
- **Poor Data Integration and Accessibility**: Data related to groundwater, climate, population, and water demand exists but is stored in silos across different agencies. These systems are not integrated or user-friendly, making it difficult for

stakeholders—especially at the local level—to access or act on the information. Visualization tools are minimal or absent.

1.4 PROPOSED SYSTEM

The proposed system leverages real-time data and satellite imagery to monitor groundwater conditions across districts. It analyses environmental parameters such as evapotranspiration (ET), soil moisture (SM), and recent rainfall trends to identify regions facing water stress. This enables accurate and up-to-date detection of scarcity levels at a granular, district-wise scale.

Using a composite water scarcity index, districts are categorized into severity zones: Low, Medium, High, and Critical. This classification allows decision-makers to quickly identify priority areas for intervention. The system dynamically adjusts these zones based on updated climate inputs and demand trends, ensuring timely and relevant assessments.

Additionally, the solution includes an interactive Streamlit dashboard, offering a clean and intuitive interface. Users can view choropleth heatmaps, real-time weather data (temperature, humidity, rainfall), and district-specific indicators all in one place. Interactive charts, tooltips, and filters enhance accessibility, even for non-technical users like local officials or farmers.

The platform delivers two core heatmaps: one visualizing groundwater availability proxies (ET + SM), and another showing the scarcity index.

1.4.1 ADVANTAGES

• **Real-Time Awareness**: Provides up-to-date information on groundwater conditions, temperature, rainfall, and humidity — allowing for faster, informed decisions compared to monthly or manual systems.

• **District Level Precision**: Offers localized insights by analyzing data at the district level, enabling more targeted interventions and better allocation of resources in high-risk regions.

• **Integrated Data Insights**: Combines satellite data, climate metrics, population growth, and water demand into one system — eliminating silos and creating a holistic view of water stress.

1.5 OBJECTIVES

- To monitor near real-time and historical environmental data using remote sensing datasets
- To develop a model that can predict future water demand based on environmental and demographic inputs.
- To classify regions based on water scarcity levels
- To visualize data through interactive maps and graphs

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

•**Software**:
The system is developed using **IDLE**, Python's default integrated development environment (IDE). IDLE provides a simple and accessible interface for writing, testing, and debugging Python scripts, making it suitable for developing both the data processing and machine learning components of the system.

•**Primary-Language**:
The project is primarily developed in **Python**, which is ideal for data analysis, machine learning, and building lightweight web applications. Libraries such as **Pandas**, **NumPy**, **GeoPandas**, **Scikit-learn**, and **Folium** are used for data manipulation, prediction modelling, and visualizations.

•**Frontend-Framework**:
The frontend is built using **Streamlit**, a popular Python library for creating interactive web applications. Streamlit is used to develop the user interface, allowing users to interact with the system and receive personalized recommendations based on their data inputs.

• **Backend Framework**:

The backend logic is also written in **Python**, using script-based processing through IDLE. It handles data merging, ML model training/inference, and real-time integration with weather APIs like OpenWeatherMap. All logic runs within the Streamlit app, making it simple yet powerful.

• **Frontend Technologies**:

The user interface is powered by Python-driven web tools, supported by:

- **HTML**: For structuring content within Streamlit.
- **CSS**: For styling and layout via Streamlit's built-in theming.
- **JavaScript**: Leveraged indirectly through **Folium** and **Leaflet.js** to enable interactive maps.
- **Plotly**: For responsive, animated charts showing trends and predictions.
- **Branca**: Used to create custom map legends and overlays.

1.6.2 HARDWARE REQUIREMENTS

Operating-System:

The system is designed to run on **Windows** and **Mac** operating systems, providing compatibility with all the development tools used in the project.

Processor:

A processor with at least an **Intel i5** or higher is recommended to handle data processing, model execution, and real-time recommendations efficiently.

RAM:

A minimum of **8GB RAM** is required to efficiently manage the system's real-time data inputs, model inference, and multi-tasking requirements. For better performance, higher RAM capacity is recommended.

2. LITERATURE SURVEY

Abdessamad Elmotawakkil et al., 2024, explores the use of Gradient Boosting Regressor (GBR), Support Vector Regression (SVR), Random Forest (RF), and Decision Tree models for groundwater prediction using satellite datasets like GRACE, MODIS, and CHIRPS. The paper achieved an impressive R^2 of 99% using GBR, showing high predictive accuracy. However, the approach is computationally intensive, with GBR requiring extensive hyperparameter tuning. A key research gap is the lack of real-time weather and population data integration, and the study is limited to Morocco [1].

Tao et al., this work utilizes Support Vector Regression (SVR) models trained on soil moisture and NDVI to forecast groundwater levels. The model is praised for effectively capturing nonlinear relationships and providing robust predictions. However, SVR's sensitivity to kernel selection and tuning affects its performance consistency. Additionally, the model's scalability to diverse terrains is limited, posing a challenge for generalized application [2].

Awan et al., this study applies ensemble learning using GBR and Random Forest models, with input features such as MODIS NDVI, Land Surface Temperature (LST), and evapotranspiration. The method delivers high precision in estimating water demand and mapping vegetation stress. Its limitation lies in the computational complexity and lack of demographic factors in the prediction model. This highlights a gap in localized human-related inputs that could enhance demand estimation accuracy [3].

Sarkar et al., proposes a climate-aware prediction approach combining GBR and weather-based models to assess future water demand. While it effectively addresses region-specific modeling and integrates climate trends, the model suffers from potential overfitting and limited spatial mapping capabilities. Crucially, it does not incorporate real-time data or consider redistribution strategies, reducing its practicality for adaptive water management [4].

Table 2.1 Literature Survey of Smart Water Analysis

Ref	Author& year of publications	Journal / Conference	Method / Approach	Merits	Limitations	Research Gap
[1]	Abdessamad Elmotawakkil et al., 2024	Journal of Hydroinformatics, IWA Publishing	Gradient Boosting Regressor (GBR), SVR, Random Forest (RF), Decision Tree (DT) using GRACE, MODIS, and CHIRPS satellite data	Achieved 99% R^2 using GBR; integrated remote sensing & ML; high predictive accuracy	Computationally intensive; GBR model needs fine hyperparameter tuning	Lacks integration of real-time weather & population data; limited to Morocco region
[2]	Tao et al., 2022	Environmental Earth Sciences, Springer	Support Vector Regression (SVR) using soil moisture and NDVI for groundwater prediction	Effective handling of nonlinear relationships; robust prediction	SVR peraformance sensitive to kernel and tuning	Limited scalability to different terrains
[3]	Awan et al., 2024	Journal of Environmental Management, Elsevier	Ensemble learning with GBR and Random Forest using MODIS NDVI, LST, evapotranspiration	High precision in water demand estimation and vegetation stress mapping	Model complexity & computational cost	Limited use of local demographic inputs
[4]	Sarkar et al., 2024	Sustainable Water Resources Management, Springer	Climate-aware groundwater demand prediction using GBR and weather models	Region-specific modeling ;integrates climate trends	Model overfitting risks; less focus on spatial mapping	Does not address real-time data use or regional redistribution

3. ANALYSIS AND DESIGN

The project addresses the challenge of predicting water scarcity at a district level using real-time and historical data. We analyzed key inputs such as groundwater levels, rainfall, temperature, and population trends. Existing systems lacked predictive capabilities and localized insights. Our solution focuses on integrating these data sources into a unified, accessible platform.

The system is designed as a modular pipeline: user input, data collection, preprocessing, machine learning prediction, and visualization. Each module is independently manageable, improving maintainability and scalability. Weather APIs and satellite data sources feed the models. The design supports both real-time and forecasted outputs through an interactive dashboard.

The frontend is designed in Streamlit, ensuring ease of use and responsiveness across devices. Choropleth maps, line charts, and dynamic tooltips are used to visualize complex data clearly. District selectors and filter options enhance interactivity. The overall design prioritizes clarity, accuracy, and user accessibility for both experts and local planners.

3.1 MODULES

Pre-processing and Feature Engineering Module

This module prepares raw input data for modelling by cleaning, normalizing, and structuring it. It handles missing values, aligns temporal data, and aggregates spatial information.

- **Input:** Raw datasets (climate, groundwater, population)
- **Output:** Cleaned, feature-engineered dataset

Machine Learning and Prediction Module

Uses algorithms like Gradient Boosting Regressor (GBR) to learn from historical data and predict groundwater levels and water demand. Trained models generate district-wise forecasts.

- **Input:** Feature-engineered dataset
- **Output:** Forecasted groundwater levels and water demand

Scarcity Classification Module

Compares predicted groundwater availability against water demand to determine severity. Regions are categorized into Very Safe, Safe, Moderate, Warning, Critical, or Severe zones.

- **Input:** Predicted levels and demand
- **Output:** Scarcity severity classification per district

Visualization and Mapping Module

Displays results via interactive choropleth maps and time-series charts using Streamlit and Folium. Makes data accessible and interpretable for decision-makers and the public.

- **Input:** Classified results and predictions
- **Output:** Interactive maps and forecast charts

3.2 ARCHITECTURE

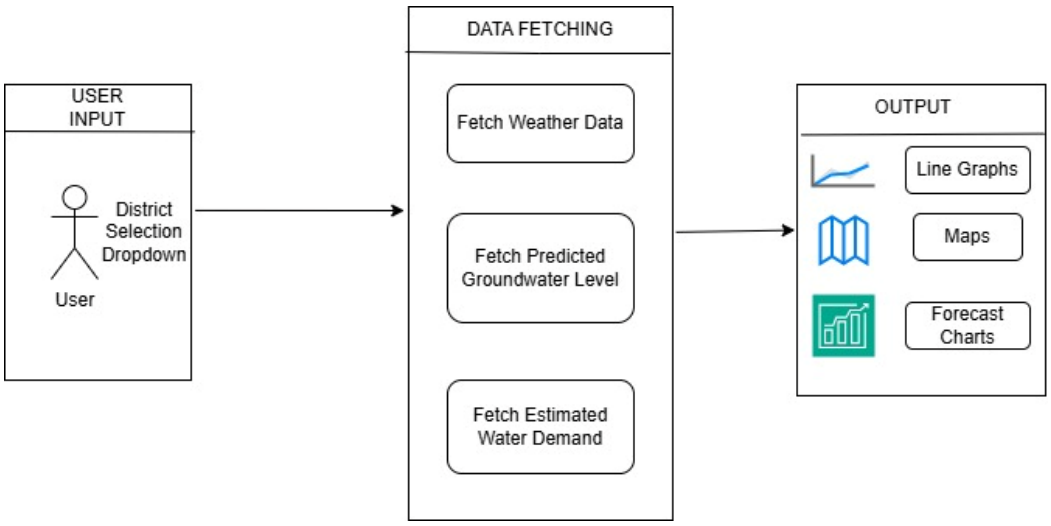


Fig. 3.2.1 Architecture of Smart Water Analysis

The architecture of the system, as shown in Fig. 3.2.1, represents a modular and user-centric design tailored for district-level water monitoring and prediction. It begins with user input and proceeds through a structured data fetching pipeline, ending with visual outputs. This flow ensures ease of use while maintaining technical robustness. The system is built to support real-time decision-making for water management.

At the entry point of the system, users interact via a district selection dropdown. This user interface allows individuals, administrators, or planners to choose a specific region of interest. The input is straightforward but essential, as it determines which district's data will be processed. This personalized selection enhances user engagement and ensures localized results.

Once the district is selected, the system activates the Data Fetching layer. This module pulls relevant information including weather data (temperature, humidity, rainfall), predicted groundwater levels (via trained machine learning models), and estimated water demand based on population and climate inputs. These steps ensure that the fetched data is both recent and specific to the selected district.

While not explicitly shown in the visual, the backend processes raw fetched data for modeling and alignment. Groundwater levels are predicted using regression models, and water demand is estimated using population and climatic conditions. Weather data enhances context and supports forecasting accuracy. The processed data is passed to the visualization layer for meaningful display.

The final layer provides output in the form of line graphs, geospatial maps, and forecast charts. Line graphs show historical and predicted groundwater levels, while maps display scarcity zones or demand intensity. Forecast charts combine multiple variables to give a future outlook. These outputs make the system highly informative, user-friendly, and suitable for actionable insights.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig.3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

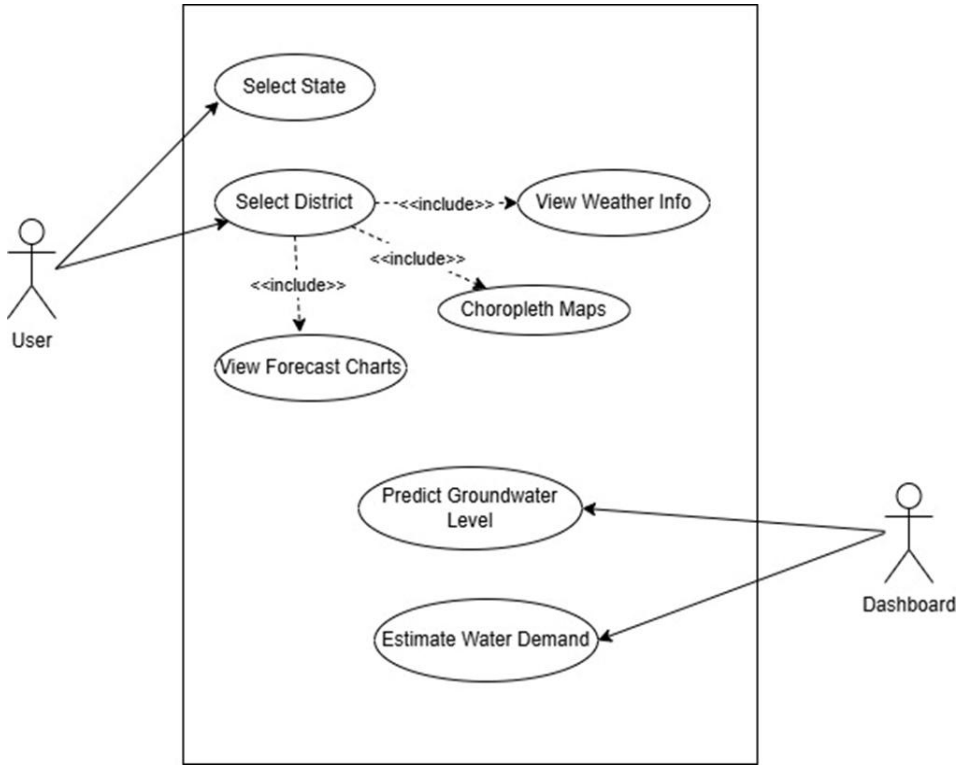


Fig.3.3.1.1 Use Case Diagram

Actors:

1. **User**: The individual interacting with the system, choosing the state and district of choice.
2. **System**: The backend system that processes user data, trains models, and generates predictions based on the user's inputs.

Use Cases:

1. **Select State:** The user chooses a specific state to begin narrowing down their area of interest for water analysis.
2. **Select District:** After selecting the state, the user selects a district to retrieve district-specific data such as groundwater, weather, and demand.
3. **View Weather Info:** The system fetches and displays real-time weather information (rainfall, temperature, humidity) for the selected district.

4. **View Choropleth Maps:** The user can view choropleth maps that visually represent groundwater levels or water scarcity status across districts.
5. **View Forecast Charts:** The system provides time-series charts forecasting groundwater availability and water demand.
6. **Predict Groundwater Level:** Using historical data and ML models, the system predicts future groundwater levels for the selected district.
7. **Estimate Water Demand:** The system estimates future water demand based on crop data, climate, and population growth trends.
8. **Dashboard Interaction:** The system's dashboard component gathers and visually presents all the outputs (weather, predictions, maps, demand, etc.) in a user-friendly interface.

3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig.3.3.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

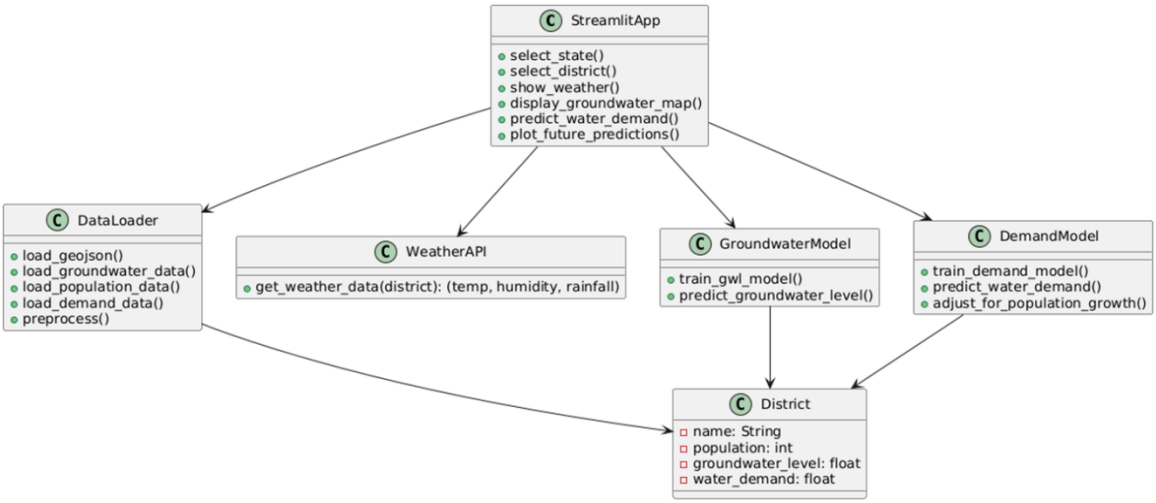


Fig.3.3.2.1 Class Diagram

Relationships:

1. **User → Streamlit Server** :
    - The User interacts with the system through the Streamlit UI**.**
    - The server handles selections (e.g., state, district), requests for data, and triggers backend logic..
2. **Streamlit Web Server → DataLoader**:
    - The web server invokes the DataLoader to fetch district-level groundwater, population, and demand data..
    - It also calls DataLoader to load GeoJSON boundaries for mapping.
3. **Streamlit Web Server → Weather API**:
    - It requests live weather information (temperature, humidity, rainfall) using OpenWeatherMap API.
    - The API response is used to provide real-time context to groundwater/demand predictions.
4. **Streamlit Web Server → Models**:
    - It calls the Groundwater Model to train and predict future groundwater levels using district-wise climate and historical level data.
    - It calls Demand Model to predict current and future water demand using population, soil recharge factor, and predicted groundwater levels.
    - The model uses Gradient Boosting Regressor (GBR) to learn from past groundwater data.
    - Similarly, GBR is used to model water demand based on climatic and groundwater inputs.
5. **Streamlit Web Server → Visualization Module**:
    - After predictions, the server triggers visualization logic to create choropleth maps and time-series line graphs.
    - This module presents current status and future projections in a user-friendly manner
6. **Visualization Module → Output**:
    - It renders choropleth maps (for scarcity and demand) and line charts (for trends over months).
    - These visualizations support comparison between districts and timeline tracking.

System Flow:

1. **User Interaction**:
    - The user selects the state and district using the Streamlit UI.
    - The backend fetches and processes corresponding data, including live weather.
2. **Data Fetching**:
    - The Data Loader loads historical groundwater, demand, and population CSVs.
    - GeoJSON files are loaded for district boundaries.
    - Live weather is fetched via OpenWeatherMap API.
3. **Prediction**:
    - The Groundwater Model uses GBR to predict groundwater level for each district.
    - The Demand Model predicts water demand using population, climate, and groundwater data, adjusted for population growth.
4. **Visualization**:
    - Choropleth map showing current groundwater level classification.
    - Line chart showing future groundwater level trend.
    - Line chart showing adjusted water demand over months.

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig.3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

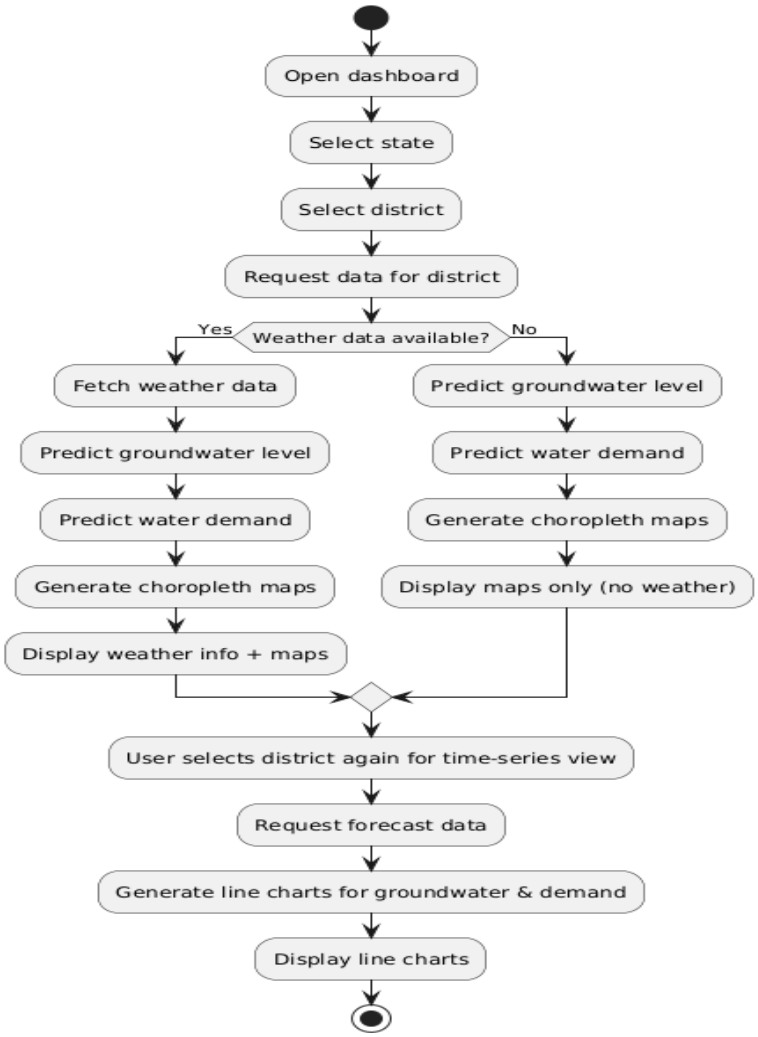


Fig.3.3.3.1 Activity Diagram

Flow Explanation:

1. **Open Dashboard**

   The user launches the Groundwater Monitoring dashboard
2. **Select State**

   The user chooses a state (e.g., Telangana or Andhra Pradesh) from the dropdown menu.
3. **Select State**

   The user chooses a district of the selected state from the dropdown menu.
4. **Request Data for District**

   The system begins processing and retrieving data (groundwater levels, population, water demand) for the selected district.
5. **Weather Data Available?**

- If the weather data is successfully retrieved then it includes the weather data.
- If the weather data is not retrieved then it does not include the weather data.

6. **Generate Maps and Predictions**

   - **Predict Groundwater Level**: A Gradient Boosting Regressor predicts the current groundwater level using historical values.
   - **Predict Water Demand**: Demand is estimated using the predicted groundwater level and current population.
   - **Generate Choropleth Maps**: The system produces district-level color-coded maps.
   - **Display Weather Info + Maps**: Both the real-time weather and predicted groundwater/demand maps are displayed.

7. **SelectDistrict**

   The user re-selects the district to visualize future trends which enables detailed forecasting over the next 12 months.

8. **Request Forecast Data**

   The system uses trained models to predict monthly groundwater level and adjusted water demand.

9. **Generate Line Charts**

   Two line chart are generated:

   - Groundwater Level Forecast
   - Water Demand Forecast

10. **DisplayCharts**

    The charts are displayed interactively using Plotly.

11. **End of Workflow**

    The process concludes, and the user can either:

    - View another district
    - Re-run predictions or
    - Exit the application

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig.3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

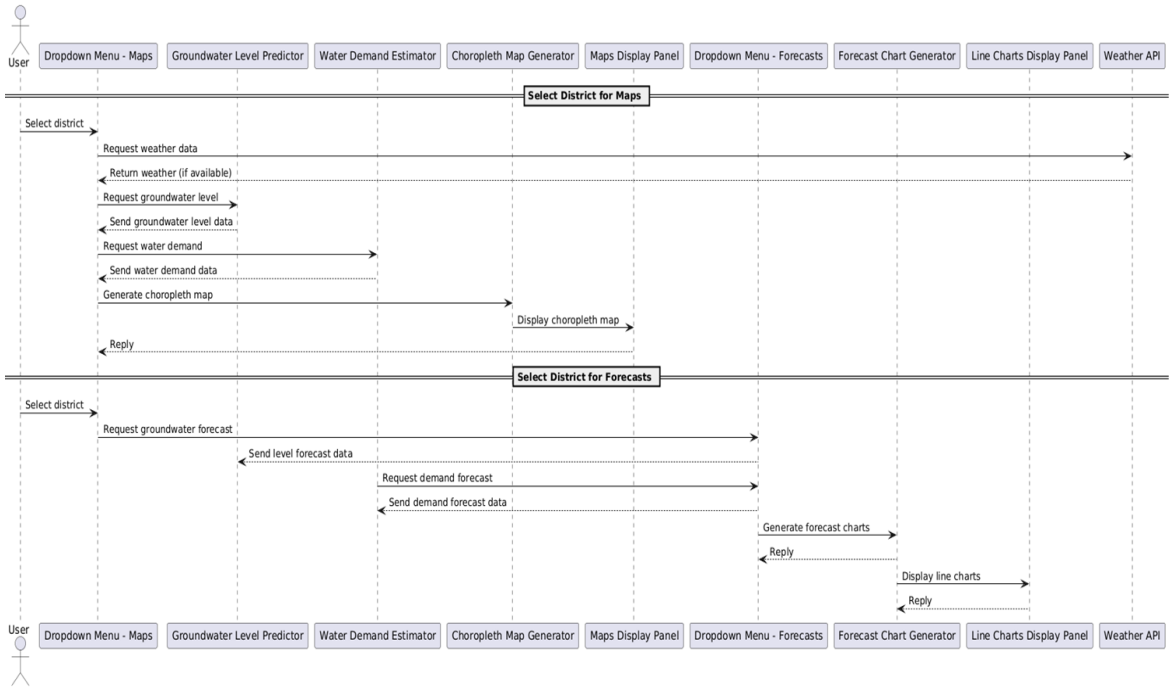


Fig.3.3.4.1 Sequence Diagram

Key Interactions and Relationships

- **User and System**:
    - **State & District Selection**: The user selects the state (e.g., Telangana or Andhra Pradesh) and district through the UI dropdown menu in the Streamlit app.
    - **Choropleth and Forecast Request**: District-level map generation and District-level forecast generation (for groundwater level and water demand)

- **System and Weather API**:
    - **Stori Fetching Real-Time Data:** When a district is selected, the system queries the OpenWeatherMap API to fetch real-time weather data (temperature, humidity, rainfall), if available.
    - **Weather Failure Fallback:** If weather data is unavailable, the system proceeds using only historical data to generate predictions**.**
- **System and Data Files**:
    - **Data Retrieval and Cleaning**: Upon district selection, the system reads relevant data files including:
    - Historical groundwater levels
    - District-wise population
    - Historical water demand
    - District boundaries in GeoJSON format
- **Predictive Models**:
    - **Data Groundwater Prediction:** Gradient Boosting Regressor is used to predict current and future groundwater levels based on district code, rainfall, temperature, and time of year.
    - **Water Demand Prediction:** A separate GBR model predicts water demand using current population, soil recharge factor, groundwater level, and environmental inputs.
    - **Population Adjustment:** Future demand is adjusted based on expected monthly population growth
- **System and User**:
    - **Displaying Groundwater Prediction:** Gradient Boosting Regressor is used to predict current and future groundwater levels based on district code, rainfall, temperature, and time of year.
    - **Water Demand Prediction:** A separate GBR model predicts water demand using current population, soil recharge factor, groundwater level, and environmental inputs.
    - **Population Adjustment:** Future demand is adjusted based on expected monthly population growth.

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig.3.3.5.1.

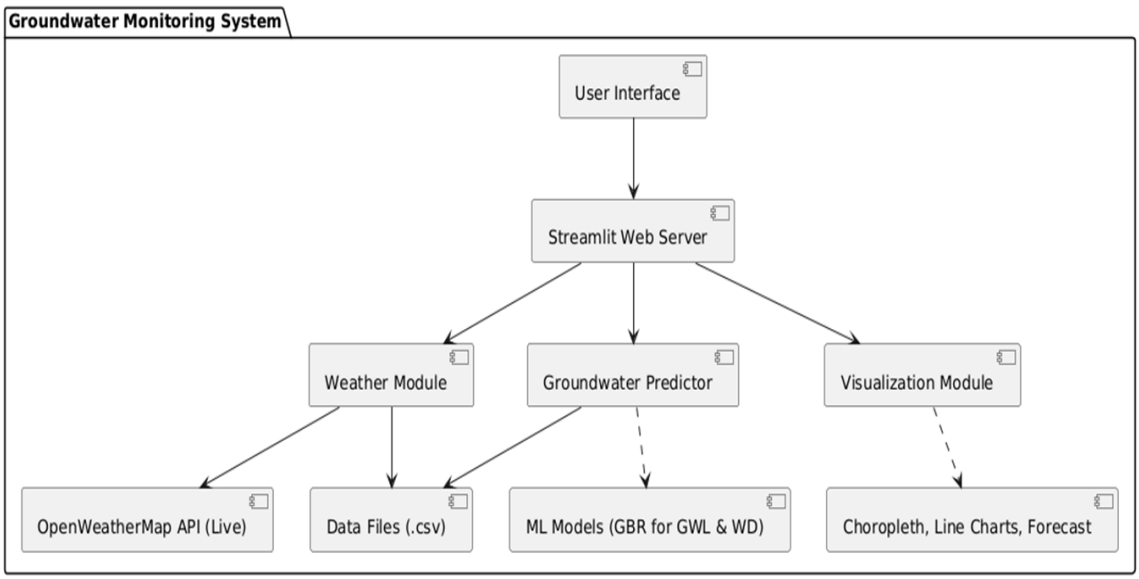


Fig.3.3.5.1 Component Diagram

Main Components:

1. **User Interface**:
    - **Description**: The front-end interface (Streamlit) through which users interact with the system. Users can select the state and district to view groundwater data, demand forecasts, real-time weather, and visual insights.

2. **Streamlit Web Server**:

- **Description**: The backend controller that manages all user interactions, triggers the predictor modules, and coordinates data fetching and chart rendering. It acts as the central brain of the system.

3. **Weather Module**:
   - **Description**: Responsible for fetching live weather data (temperature, humidity, rainfall) using the OpenWeatherMap API.
4. **Groundwater Prediction**:
   - **Description**: Performs machine learning-based predictions for groundwater levels and water demand. It utilizes district-level environmental and population data and passes results to the visualization module.
5. **Visualization Module**:
   - **Description**: Handles the creation and display of all visual outputs, including choropleth maps (for district-wise scarcity/demand) and time-series charts (forecasting groundwater and demand).
6. **OpenWeatherMap API**:
   - **Description**: An external API service used to retrieve live weather parameters for the selected district. Data includes temperature, rainfall, and humidity, which enhance prediction accuracy and dashboard information.
7. **Data Files**:
   - **Description**: Includes pre-processed or historical data for groundwater levels, population, water demand, and district geometries (GeoJSON/CSV). These serve as inputs to both model training and live predictions.
8. **ML Models**:
   - **Description**: Pre-trained **Gradient Boosting Regressor** models used for predicting groundwater level and water demand for the selected district. Models are trained on features such as rainfall, temperature, month, district code, and soil recharge factor.
9. **Choropleth, Line Charts and Forecast**:
   - **Description**: The visual outputs generated and displayed to users: **Choropleth maps**: Show scarcity and demand at district level. **Line charts**: Show monthly trends of predicted groundwater and demand. **Forecast**: Visual time series adjusted for population growth.

3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of software components across hardware nodes in the Smart Water Analysis system. It captures the interactions between the user device, servers, and API's , highlighting how system components communicate and are distributed in a real-world deployment scenario.

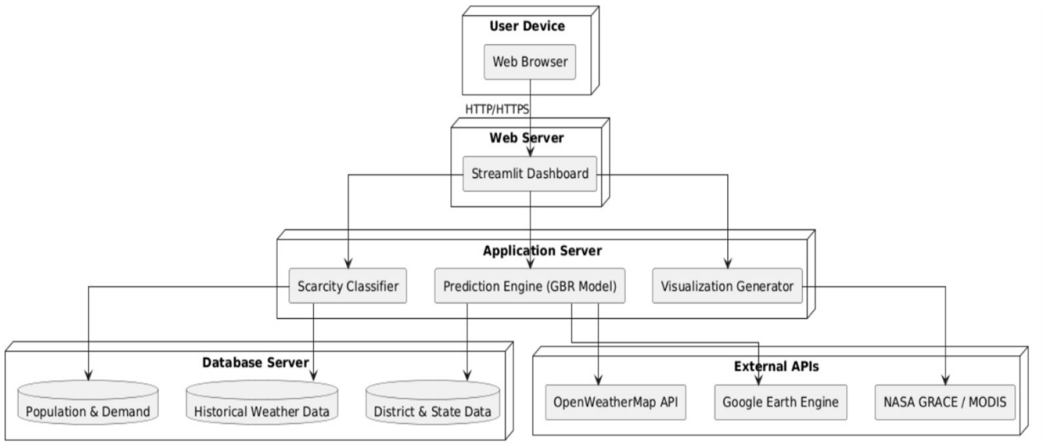


Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across three main nodes:

- **User Device:** Runs the web browser where users choose select state and district from dropdown menus. Communicates with the server via HTTP requests.
- **Web Server:** This node hosts the **Streamlit-based web application**, which acts as a lightweight frontend and request handler. It accepts user input, calls application logic (prediction engine, classifier) and triggers visualizations.
- **Application Server:** This server executes business logic and prediction models. It integrates pretrained Gradient Boosting Regressors **(GBR), p**opulation growth logic along with Soil recharge and weather correlation.
- **Database Server:** Static .csv and .geojson datasets are loaded and processed at runtime. These serve as the input for both the classifier and prediction engine.

- **External APIs:** OpenWeatherMap API Google Earth Engine NASA GRACE APIs enhance real-time predictions and enrich environmental context.

This setup ensures smooth interaction between front-end, back-end, and data layers in the Smart Water Scarcity Analysis system.

3.4 METHODOLOGY

Data Acquisition

- **Live Data**: The system integrates real-time environmental data using the OpenWeatherMap API, NASA GRACE providing up-to-date weather parameters like temperature, humidity, and rainfall for selected districts.
- **Historical Data**: Historical datasets are sourced from government portals and curated .csv files.
- **Purpose**: The historical data is used to train the models, while live data ensures that predictions remain relevant and updated.

Model Steps

- **Data Preprocessing** : All data is cleaned, encoded and aligned based on temporal granularity (monthly) for consistency across features.
- **Feature Engineering**: Variables like month number, rainfall, temperature, and soil recharge factor are constructed as model inputs. Population data is also adjusted using a fixed **growth rate** (e.g., 1.5% per year) to account for projected water demand.
- **Forecast Horizon**: The system generates **12-month forecasts**, allowing users to view upcoming trends in both groundwater levels and water demand for their selected district.

Models Used

1. **Gradient Boosting Regressor (GBR)- for Groundwater Level Prediction**

- **Description**: A powerful ensemble learning model that builds a series of decision trees to correct prediction errors sequentially. It's well-suited for complex, non-linear relationships in time-series environmental data.
- **How it Works**:
    - Inputs: District code, month number, rainfall, temperature
    - Output: Predicted groundwater level (in meters Below Ground Level)
    - The model learns from historical monthly variations and district-level attributes to forecast groundwater availability.

- **Why it's Used**: GBR is robust to overfitting and performs well even with modestly sized datasets. Its ability to capture subtle patterns in time-series makes it ideal for environmental forecasting.

2. GBR – for Water Demand Estimation

- **Description**: The same learning approach is applied to model district-level water demand. It leverages both environmental factors and population dynamics to estimate monthly demand in MLD (Million Liters per Day).
- **How it Works**:
    - Inputs: District code, rainfall, temperature, soil recharge factor, population, and predicted groundwater level
    - Output: Predicted water demand
    - Population growth is factored into monthly projections for more realistic estimates.
- **Why it's Used**: The model can handle multiple non-linear factors, like soil recharge interaction with rainfall, or the inverse relationship between groundwater depth and demand. GBR's accuracy and interpretability make it a suitable choice.

4. CODE AND IMPLEMENTATION

4.1 CODE

app.py

```
import streamlit as st
import pandas as pd
import geopandas as gpd
import plotly.express as px
import folium
from streamlit_folium import st_folium
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
import requests
from branca.element import Template, MacroElement

st.set_page_config(layout="wide")
st.title("💧 Groundwater Monitoring & Water Demand Prediction")

# --- Select State ---
state = st.selectbox("Select State", ["Telangana", "Andhra Pradesh"])

# Helper function
def classify_scarcity(level):
    if level < 2: return "Very Safe", "#00FF00"
    elif level < 5: return "Safe", "#7FFF00"
    elif level < 10: return "Moderate", "#FFFF00"
    elif level < 20: return "Warning", "#FFA500"
    elif level < 40: return "Critical", "#FF4500"
    return "Severe", "#8B0000"

# --- Telangana Block ---
if state == "Telangana":
    geojson_path = r"C:/Users/Admin/Downloads/TELANGANA_DISTRICTS.geojson"
```

```
    groundwater_path = r"C:/Users/Admin/Downloads/may_corrected (1).csv"
    population_path = r"C:/Users/Admin/Downloads/population_corrected (1).csv"
    demand_path = r"C:/Users/Admin/Downloads/demand_estimate_corrected (1).csv"
    line_chart_path = r"C:/Users/Admin/OneDrive/line chart.csv"

    df = pd.read_csv(groundwater_path)

# Optional corrections
    correction_dict = {
        'MEDCHAL': 'MEDCHAL MALKAJGIRI'
    }
    df['district'] = df['district'].replace(correction_dict)
    df['district'] = df['district'].str.upper().str.strip()
    df.rename(columns={'Static Water Level (mbgl)': 'groundwater_level'}, inplace=True)

# Read GeoJSON
    gdf = gpd.read_file(geojson_path)
    gdf['dtname'] = gdf['dtname'].str.upper().str.strip()

# Population
    pop = pd.read_csv(population_path)
    pop.rename(columns={'District': 'district', 'Population': 'population'}, inplace=True)
    pop['district'] = pop['district'].str.upper().str.strip()

# Water demand
    demand = pd.read_csv(demand_path)
    demand.rename(columns={'District': 'district', 'Estimated_Water_Demand_MLD':
'water_demand'}, inplace=True)
    demand['district'] = demand['district'].str.upper().str.strip()

# Clean groundwater data
    df = df.dropna(subset=['groundwater_level'])

# Average groundwater level per district
```

```
    df_latest = df.groupby('district', as_index=False)['groundwater_level'].mean()

# Merge with population and demand
    df_latest = df_latest.merge(pop, on='district', how='left')
    df_latest = df_latest.merge(demand, on='district', how='left')
    df_latest = df_latest.dropna()

# ☑ Correct dropdown using df_latest
    selected_district = st.selectbox("Select District", df_latest['district'].sort_values())



    # --- Weather ---
    def get_weather_data(district_name):
        api_key = "782a2f11195ba039b53da13244a2a4fd"
        params = {'q': f"{district_name},IN", 'appid': api_key, 'units': 'metric'}
        try:
            r = requests.get("http://api.openweathermap.org/data/2.5/weather",
params=params, timeout=5)
            d = r.json()
            if r.status_code == 200:
                return d['main']['temp'], d['main']['humidity'], d.get('rain', {}).get('1h', 0)
        except: return None, None, None
        return None, None, None

    temp, humidity, rainfall = get_weather_data(selected_district.title())
    st.subheader("⛈ Current Weather")
    if temp:
        st.metric("Temperature (°C)", f"{temp:.1f}")
        st.metric("Humidity (%)", f"{humidity}%")
        st.metric("Rainfall (mm)", f"{rainfall} mm")
    else:
        st.warning("Live weather unavailable.")
```

```
    # --- Groundwater Choropleth ---
    st.subheader("🗺️ Groundwater Scarcity Map")
    df_best = df.sort_values('groundwater_level').drop_duplicates('district')
    pred_levels = []
    for dist in df_best['district'].unique():
        dist_df = df[df['district'] == dist]
        X = pd.DataFrame({'index': range(len(dist_df))})
        y = dist_df['groundwater_level']
        if len(y) >= 2:
            model = GradientBoostingRegressor().fit(X, y)
            pred = model.predict([[len(dist_df)]])[0]
            pred_levels.append({'district': dist, 'predicted_level': pred})
    pred_df = pd.DataFrame(pred_levels)
    merged = gdf.merge(pred_df, left_on="dtname", right_on="district", how="inner")
    m = folium.Map(location=[17.5, 79.0], zoom_start=7)
    for _, row in merged.iterrows():
        level = row['predicted_level']
        status, color = classify_scarcity(level)
        folium.GeoJson(row['geometry'], style_function=lambda _, c=color: {"fillColor": c,
"color": "black", "weight": 1, "fillOpacity": 0.7},
                       tooltip=folium.Tooltip(f"{row['dtname'].title()}: {level:.2f} m BGL
({status})")).add_to(m)
    # Groundwater scarcity color classification logic
    color_legend = [
        ("Very Safe", "< 2 m", "#00FF00"),
        ("Safe", "2–5 m", "#7FFF00"),
        ("Moderate", "5–8 m", "#FFFF00"),
        ("Warning", "8–11 m", "#FFA500"),
        ("Critical", "11–15 m", "#FF4500"),
        ("Severe", "> 15 m", "#8B0000")]

    st.subheader("🖍️ Groundwater Level Color Legend")
    for label, range_text, color in color_legend:
```

```
        st.markdown(f"<span style='color:{color}; font-weight:bold;'>●</span> {label}:
{range_text}", unsafe_allow_html=True)

    st_folium(m, width=700)
    # --- Predicted Water Demand ---
    st.subheader("🚰 Predicted Water Demand")
    X_demand = df_latest[['population', 'groundwater_level']]
    y_demand = df_latest['water_demand']
    model = GradientBoostingRegressor().fit(X_demand, y_demand)
    df_latest['predicted_water_demand'] = model.predict(X_demand)
    if selected_district in df_latest['district'].values:
        demand_value = df_latest[df_latest['district'] ==
selected_district]['predicted_water_demand'].values[0]
        st.success(f"Predicted Water Demand for **{selected_district.title()}**:
**{demand_value:.2f} MLD**")
    else:
        st.warning("Water demand prediction not available.")
    st.subheader("🌍 Water Demand Map")
    merged_demand = gdf.merge(df_latest, left_on="dtname", right_on="district",
how="inner")
    m2 = folium.Map(location=[17.5, 79.0] if state == "Telangana" else [16.5, 80.5],
zoom_start=7, tiles="cartodbpositron")
    folium.Choropleth(
        geo_data=gdf,
        data=merged_demand,
        columns=["district", "predicted_water_demand"],
        key_on="feature.properties.dtname",
        fill_color="YlGnBu",
        fill_opacity=0.7,
        line_opacity=0.2,
        legend_name="Predicted Water Demand (MLD)"
        ).add_to(m2)
    for _, row in merged_demand.iterrows():
```

```
        folium.GeoJson(
            row["geometry"],
            style_function=lambda feature: {
                "fillOpacity": 0,
                "color": "black",
                "weight": 0.5
            },
            tooltip=folium.Tooltip(
                f"{row['district'].title()}<br>Demand: {row['predicted_water_demand']:.2f}
MLD"
            )
        ).add_to(m2)
    st_folium(m2, width=700)
    import pandas as pd
    import plotly.express as px
    from sklearn.ensemble import GradientBoostingRegressor
    from sklearn.preprocessing import LabelEncoder

# Load your real past groundwater + climate data
    past_data = pd.read_csv(r"C:\Users\Admin\OneDrive\line chart.csv")

# Load current water demand per district
    demand_data = pd.read_csv(r"C:\Users\Admin\Downloads\demand_estimate_corrected
(1).csv")

# Load current population per district
    population_data = pd.read_csv(r"C:\Users\Admin\Downloads\population_corrected
(1).csv")

# Prepare month list
    future_months = ['January', 'February', 'March', 'April', 'May', 'June','July', 'August',
'September', 'October', 'November', 'December']

# Create mapping for months to numbers
```

```
    month_mapping = {month: idx for idx, month in enumerate(future_months, start=1)}
    past_data['Month_Num'] = past_data['Month_Num'].map(month_mapping)

    past_data['Soil_Recharge_Factor'] = past_data['Soil_Recharge_Factor']

# Merge demand estimates into past_data
    past_data = past_data.merge(demand_data[['District', 'water_demand']], on='District',
how='left')
    past_data = past_data.dropna(subset=['water_demand'])

# Merge population estimates into past_data
    past_data = past_data.merge(population_data[['District', 'Population']], on='District',
how='left')

# Label encode District
    le = LabelEncoder()
    past_data['District_Code'] = le.fit_transform(past_data['District'])

# --- Groundwater Level Model ---

    X_gwl = past_data[['District_Code', 'Month_Num', 'Rainfall_mm', 'Temperature_C']]
    y_gwl = past_data['Groundwater_Level_mBGL']
    model_gwl = GradientBoostingRegressor().fit(X_gwl, y_gwl)

# --- Water Demand Model ---

    X_demand = past_data[['District_Code', 'Month_Num', 'Rainfall_mm', 'Temperature_C',
'Soil_Recharge_Factor', 'Groundwater_Level_mBGL']]
    y_demand = past_data['water_demand']
    model_demand = GradientBoostingRegressor().fit(X_demand, y_demand)
    average_rainfall = past_data['Rainfall_mm'].mean()
    average_temp = past_data['Temperature_C'].mean()

# Future Month to Year mapping (for population growth adjustment)
```

```
    months_to_years = {
        'January': 0,
        'February': 1/12,
        'March': 2/12,
        'April': 3/12,
        'May': 4/12,
        'June': 5/12,
        'July': 6/12,
        'August': 7/12,
        'September': 8/12,
        'October': 9/12,
        'November': 10/12,
        'December': 11/12}
    population_growth_rate = 0.015  # 1.5% annual growth

# Generate future predictions for each district separately
    future_predictions = []
    for district in past_data['District'].unique():
        district_code = le.transform([district])[0]
        soil_factor = past_data[past_data['District'] ==
district]['Soil_Recharge_Factor'].iloc[0]
        current_population = past_data[past_data['District'] == district]['Population'].iloc[0]
        future_df = pd.DataFrame({
            'District': [district]*12,
            'District_Code': [district_code]*12,
            'Month': future_months,
            'Month_Num': list(range(1, 13)),
            'Rainfall_mm': [average_rainfall]*12,
            'Temperature_C': [average_temp]*12,
            'Soil_Recharge_Factor': [soil_factor]*12,
            'population': [current_population]*12})

        X_future_gwl = future_df[['District_Code', 'Month_Num', 'Rainfall_mm',
'Temperature_C']]
```

```
        future_df['Predicted_Groundwater_Level_mBGL'] =
model_gwl.predict(X_future_gwl)

    # Predict water demand
        X_future_demand = future_df[['District_Code', 'Month_Num', 'Rainfall_mm',
'Temperature_C', 'Soil_Recharge_Factor', 'Predicted_Groundwater_Level_mBGL']]
        X_future_demand =
X_future_demand.rename(columns={'Predicted_Groundwater_Level_mBGL':
'Groundwater_Level_mBGL'})
        future_df['Predicted_Water_Demand_MLD'] =
model_demand.predict(X_future_demand)
        future_df['Years_into_Future'] = future_df['Month'].map(months_to_years)
        future_df['Future_Population'] = future_df['population'] * ((1 +
population_growth_rate) ** future_df['Years_into_Future'])
        future_df['Adjusted_Predicted_Water_Demand_MLD'] =
future_df['Predicted_Water_Demand_MLD'] * (future_df['Future_Population'] /
future_df['population'])
        future_predictions.append(future_df)
    final_future_predictions = pd.concat(future_predictions, ignore_index=True)

# --- 📈 Plotting Section ---

    st.subheader("📍 Select District to View Future Predictions")
    selected_district = st.selectbox("Select District",
final_future_predictions['District'].unique())

    selected_future_df = final_future_predictions[final_future_predictions['District'] ==
selected_district]

# --- 📈 Plot 1: Groundwater Level Line Chart
    st.subheader(f"📈 Predicted Future Groundwater Level - {selected_district}")

    fig_gwl = px.line(
```

```
        selected_future_df,
        x='Month',
        y='Predicted_Groundwater_Level_mBGL',
        markers=True,
        labels={'Predicted_Groundwater_Level_mBGL': 'Groundwater Level (m BGL)'},
        title=f'Groundwater Level Prediction for {selected_district}')
    st.plotly_chart(fig_gwl, use_container_width=True)

# --- 📈 Plot 2: Adjusted Water Demand Line Chart
    st.subheader(f"📈 Predicted Future Water Demand (Adjusted for Population Growth) -
{selected_district}")

    fig_demand = px.line(
        selected_future_df,
        x='Month',
        y='Adjusted_Predicted_Water_Demand_MLD',
        markers=True,
        labels={'Adjusted_Predicted_Water_Demand_MLD': 'Water Demand (MLD)'},
        title=f'Water Demand Prediction for {selected_district}')
    st.plotly_chart(fig_demand, use_container_width=True)

# --- End of Future Prediction Section ---


else:# --- Andhra Pradesh Block ---
    geojson_path = r"C:/Users/Admin/Downloads/andhra-pradesh.geojson"
    groundwater_path = r"C:/Users/Admin/Downloads/reshaped_groundwater_levels.csv"
    population_path =
r"C:/Users/Admin/Downloads/andhra_pradesh_district_population.csv"
    demand_path =
r"C:/Users/Admin/Downloads/andhra_pradesh_water_demand_mld.csv"

    # Load datasets
```

```
    df = pd.read_csv(groundwater_path)
    pop = pd.read_csv(population_path)
    demand = pd.read_csv(demand_path)
    gdf = gpd.read_file(geojson_path)

    # Clean column names
    df.columns = df.columns.str.strip().str.lower()
    pop.columns = pop.columns.str.strip()
    demand.columns = demand.columns.str.strip()
    gdf.columns = gdf.columns.str.strip()

    # Rename 'District' to 'district' consistently
    pop.rename(columns={'District': 'district', 'Population': 'population'}, inplace=True)
    demand.rename(columns={'District': 'district', 'Estimated_Water_Demand_MLD':
'water_demand'}, inplace=True)

    # Standardize and clean all district names
    district_rename_map = {
        "ANANTHAPURAMU": "ANANTAPUR",
        "ANANTAPURAMU": "ANANTAPUR",
        "YSR KADAPA": "YSR",
        "Y.S.R KADAPA": "YSR",
        "Y.S.R. KADAPA": "YSR",
        "SRI POTTI SRIRAMULU NELLORE": "NELLORE",
        "DR. B. R. AMBEDKAR KONASEEMA": "KONASEEMA",
        "SRI SATHYA SAI": "SATHYA SAI",
        "ALLURI SITHARAMA RAJU": "ALLURI SITARAMA RAJU",
        "BAPATLA": "BAPATLA",
        "ANNAMAYYA": "ANNAMAYYA",
        "NTR": "NTR",
        "ANAKAPALLI": "ANAKAPALLI"
    }

    for df_to_fix in [df, pop, demand]:
```

```
        df_to_fix['district'] =
df_to_fix['district'].str.upper().str.strip().replace(district_rename_map)
        df_to_fix.sort_values('district', inplace=True)

    gdf['dtname'] = gdf['name'].str.upper().str.strip().replace(district_rename_map)

    # Drop rows missing groundwater levels
    df = df.dropna(subset=['groundwater_level'])

    # Merge all on 'district'
    df_latest = df.groupby('district', as_index=False)['groundwater_level'].mean()
    df_latest = df_latest.merge(pop, on='district', how='inner')
    df_latest = df_latest.merge(demand, on='district', how='inner')

    # Check if merged dataset is valid
    if df_latest.empty:
        st.error("❌ Merged data is empty. Check if district names align across files.")
        st.write("Groundwater districts:", sorted(df['district'].unique()))
        st.write("Population districts:", sorted(pop['district'].unique()))
        st.write("Demand districts:", sorted(demand['district'].unique()))
    else:
        selected_district = st.selectbox("Select District", sorted(df_latest['district'].unique()))

        # --- Groundwater Scarcity Map ---
        st.subheader("🗺 Groundwater Scarcity Map")
        df_best = df.sort_values('groundwater_level').drop_duplicates('district')
        pred_levels = []
        for dist in df_best['district'].unique():
            dist_df = df[df['district'] == dist]
            X = pd.DataFrame({'index': range(len(dist_df))})
            y = dist_df['groundwater_level']
            if len(y) >= 2:
                model = GradientBoostingRegressor().fit(X, y)
```

```
            pred = model.predict([[len(dist_df)]])[0]
            pred_levels.append({'district': dist, 'predicted_level': pred})
    pred_df = pd.DataFrame(pred_levels)
    merged = gdf.merge(pred_df, left_on="dtname", right_on="district", how="inner")
    m = folium.Map(location=[16.5, 80.5], zoom_start=7)
    for _, row in merged.iterrows():
        level = row['predicted_level']
        status, color = classify_scarcity(level)
        folium.GeoJson(row['geometry'], style_function=lambda _, c=color: {
            "fillColor": c, "color": "black", "weight": 1, "fillOpacity": 0.7
        }, tooltip=folium.Tooltip(f"{row['dtname'].title()}: {level:.2f} m BGL
({status})")).add_to(m)
    color_legend = [
    ("Very Safe", "< 2 m", "#00FF00"),
    ("Safe", "2–5 m", "#7FFF00"),
    ("Moderate", "5–8 m", "#FFFF00"),
    ("Warning", "8–11 m", "#FFA500"),
    ("Critical", "11–15 m", "#FF4500"),
    ("Severe", "> 15 m", "#8B0000")]
    st.subheader("✏️ Groundwater Level Color Legend")
    for label, range_text, color in color_legend:
        st.markdown(f"<span style='color:{color}; font-weight:bold;'>●</span> {label}:
{range_text}", unsafe_allow_html=True)
    st_folium(m, width=700)

    # --- Predicted Water Demand ---
    st.subheader("🚰 Predicted Water Demand")
    X_demand = df_latest[['population', 'groundwater_level']]
    y_demand = df_latest['water_demand']
    model = GradientBoostingRegressor().fit(X_demand, y_demand)
    df_latest['predicted_water_demand'] = model.predict(X_demand)
    if selected_district in df_latest['district'].values:
```

```
        demand_value = df_latest[df_latest['district'] ==
selected_district]['predicted_water_demand'].values[0]
        st.success(f"Predicted Water Demand for **{selected_district.title()}**:
**{demand_value:.2f} MLD**")
    else:
        st.warning("Water demand prediction not available.")

    # --- Choropleth Map - Water Demand ---
    st.subheader("🌍 Water Demand Map")
    merged_demand = gdf.merge(df_latest, left_on="dtname", right_on="district",
how="inner")
    m2 = folium.Map(location=[16.5, 80.5], zoom_start=7)
    folium.Choropleth(
        geo_data=gdf,
        data=merged_demand,
        columns=["district", "predicted_water_demand"],
        key_on="feature.properties.dtname",
        fill_color="YlGnBu",
        fill_opacity=0.7,
        line_opacity=0.2,
        legend_name="Predicted Water Demand (MLD)"
    ).add_to(m2)
    for _, row in merged_demand.iterrows():
        folium.GeoJson(
            row["geometry"],
            style_function=lambda feature: {
                "fillOpacity": 0,
                "color": "black",
                "weight": 0.5
            },
            tooltip=folium.Tooltip(
                f"{row['district'].title()}<br>Demand: {row['predicted_water_demand']:.2f}
MLD"
```

```
            )
        ).add_to(m2)
    st_folium(m2, width=700)
    le_ap = LabelEncoder()
    df_ap =
pd.read_csv("C:/Users/Admin/Downloads/andhra_pradesh_groundwater_merged.csv")

# Standardize and clean district names
    df_ap['District'] = df_ap['district'].str.upper().str.strip()
    district_rename_map = {
        "ANANTHAPURAMU": "ANANTAPUR",
        "ANANTAPURAMU": "ANANTAPUR",
        "YSR KADAPA": "YSR",
        "Y.S.R KADAPA": "YSR",
        "Y.S.R. KADAPA": "YSR",
        "SRI POTTI SRIRAMULU NELLORE": "NELLORE",
        "DR. B. R. AMBEDKAR KONASEEMA": "KONASEEMA",
        "SRI SATHYA SAI": "SATHYA SAI",
        "ALLURI SITHARAMA RAJU": "ALLURI SITARAMA RAJU",
        "BAPATLA": "BAPATLA",
        "ANNAMAYYA": "ANNAMAYYA",
        "NTR": "NTR",
        "ANAKAPALLI": "ANAKAPALLI"}
    df_ap['District'] = df_ap['District'].replace(district_rename_map)

# Convert Month Name to Numeric
    month_mapping = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,'Jul': 7, 'Aug':
8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
    df_ap['Month_Num'] = df_ap['Month_Num'].map(month_mapping)

# Drop missing values after mapping
    required_cols = ['District', 'Month_Num', 'Rainfall_mm',
'Temperature_C','Groundwater_Level_mBGL', 'water_demand', 'Soil_Recharge_Factor',
'Population']
```

```
    df_ap = df_ap.dropna(subset=required_cols)
    df_ap['District_Code'] = le_ap.fit_transform(df_ap['District'].astype(str))

# --- Train Groundwater Level Model ---
    X_ap_gwl = df_ap[['District_Code', 'Month_Num', 'Rainfall_mm', 'Temperature_C']]
    y_ap_gwl = df_ap['Groundwater_Level_mBGL']
    model_ap_gwl = GradientBoostingRegressor().fit(X_ap_gwl, y_ap_gwl)

# --- Train Water Demand Model ---
    X_ap_demand = df_ap[['District_Code', 'Month_Num', 'Rainfall_mm',
'Temperature_C','Soil_Recharge_Factor', 'Groundwater_Level_mBGL']]
    y_ap_demand = df_ap['water_demand']
    model_ap_demand = GradientBoostingRegressor().fit(X_ap_demand, y_ap_demand)

# --- Forecast for Selected District ---
    selected_code = le_ap.transform([selected_district])[0]
    district_data = df_ap[df_ap['District'] == selected_district]
    if not district_data.empty:
        avg_rain = df_ap['Rainfall_mm'].mean()
        avg_temp = df_ap['Temperature_C'].mean()
        pop_growth = 0.015  # 1.5%
        base_population = district_data['Population'].iloc[0]
        soil_factor = district_data['Soil_Recharge_Factor'].iloc[0]
        future_months = list(range(1, 13))
        month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun','Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec']
        future_df = pd.DataFrame({
            'Month_Num': future_months,
            'Month': month_names,
            'District_Code': selected_code,
            'Rainfall_mm': avg_rain,
            'Temperature_C': avg_temp,
            'Soil_Recharge_Factor': soil_factor})
```

```
    # Predict groundwater levels
        X_future_gwl = future_df[['District_Code', 'Month_Num', 'Rainfall_mm',
'Temperature_C']]
        future_df['Predicted_Groundwater_Level_mBGL'] =
model_ap_gwl.predict(X_future_gwl)

    # Adjusted population growth per month
        future_df['Years_into_Future'] = future_df['Month_Num'] / 12
        future_df['Future_Population'] = base_population * ((1 + pop_growth) **
future_df['Years_into_Future'])

    # Predict water demand
        X_future_demand = future_df.copy()
        X_future_demand['Groundwater_Level_mBGL'] =
future_df['Predicted_Groundwater_Level_mBGL']
        X_future_demand = X_future_demand[['District_Code', 'Month_Num',
'Rainfall_mm', 'Temperature_C','Soil_Recharge_Factor', 'Groundwater_Level_mBGL']]
        future_df['Predicted_Water_Demand_MLD'] =
model_ap_demand.predict(X_future_demand)

    # Adjust water demand based on population growth
        future_df['Adjusted_Water_Demand_MLD'] =
future_df['Predicted_Water_Demand_MLD'] * (future_df['Future_Population'] /
base_population)

    # --- Plot Groundwater Level Forecast ---
        st.subheader(f"🗺️ Groundwater Level Forecast – {selected_district.title()}")
        fig_gwl = px.line(
            future_df,
            x='Month',
            y='Predicted_Groundwater_Level_mBGL',
            markers=True,
```

```
            labels={'Predicted_Groundwater_Level_mBGL': 'Groundwater Level (m
BGL)'})
        st.plotly_chart(fig_gwl, use_container_width=True)

    # --- Plot Water Demand Forecast ---
        st.subheader(f"📈 Water Demand Forecast – {selected_district.title()}")
        fig_demand = px.line(
            future_df,
            x='Month',
            y='Adjusted_Water_Demand_MLD',
            markers=True,
            labels={'Adjusted_Water_Demand_MLD': 'Water Demand (MLD)'})
        st.plotly_chart(fig_demand, use_container_width=True)
    else:
        st.warning(f"No valid data found for {selected_district}")
```

4.2 IMPLEMENTATION

Installing Python Packages

Open your terminal and run the following command to install all required packages:

pip install streamlit pandas scikit-learn geopandas plotly folium requests branca

Setup Google Earth Engine and authenticate using-

```
import ee
ee.Authenticate()
ee.Initialize()
```

Optional: If you're using VS Code, make sure your virtual environment is activated before installing.

Data Acquistion

- Download Static groundwater data from WRIS, Open Data Telangana,etc.
- Download the population and water demand data.
- Use OpenWeatherAPI to fetch live weather data
- Choose Datasets like NASA MODIS or EOS04 from Bhoonidhi and use Google Earth Engine API to extract that data.
- Export and save the values to .csv
- Then write rest all of remaining code handling null values and mismatched names as well.

Running the Application

1. **Navigate to the project root directory in Windows PowerShell:**
   cd Admin
2. **Start the Streamlit Server:**
   streamlit run app.py
3. **Open your browser and go to:**
   localhost:3000

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is an essential phase in software development that verifies the correct functionality, usability, performance, and reliability of the application under varying conditions. It ensures the final system meets the desired specifications and behaves consistently under expected and unexpected inputs. Effective testing minimizes the risk of system failures and enhances the user experience by identifying bugs or inconsistencies early in the lifecycle.

In this project, Groundwater Monitoring & Water Demand Prediction, testing was crucial to ensure that real-time weather integration, district-wise forecasting, data-driven visualizations, and user interaction flows all functioned as intended. The goal was to validate that each module—from district selection to forecast generation and choropleth rendering—performed seamlessly and accurately contributed to groundwater scarcity assessment and resource planning.

The test cases were crafted to verify the major functionalities and flow of the application, including:

- Correct transition between pages and components such as state and district dropdowns and switching between dashboard and forecast view
- Accurate generation of district-level groundwater and water demand predictions using Gradient Boosting Regressor models
- Proper display and classification of groundwater scarcity through choropleth maps with dynamic color-coded legends
- Reliable integration and fallback handling of OpenWeatherMap API for temperature, humidity, and rainfall data
- Dynamic time-series forecasting using historical data and seasonal variables for monthly groundwater and demand predictions

5.2 TEST CASES:

Table 5.1 Test Cases of Smart Water Analysis

Test Case ID	Test case Name	Test Description	Expected Output	Actual Output	Remarks
TC_01	State Selection	Select between Telangana and Andhra Pradesh	State and district data load correctly for selected state	Data loaded correctly for selected state	Success
TC_02	District Selection	Select district from dropdown	Corresponding data loads	District dropdown and corresponding data loads	Success
TC_03	Weather API Integration	Fetch live weather data for selected district	Weather data for selected district displays	Weather data for selected district displayed	Success
TC_04	Groundwater Scarcity Map	Generate and display choropleth map	Districts shaded by scarcity level , legend shown	Map renders with legend and shaded districts	Success
TC_05	Predicted Water Demand Calculation	Predict water demand using population and groundwater level	Demand in MLD shown	Demand in MLD shown	Success
TC_06	Future Water Demand Prediction	Forecast adjusted water demand based on population growth	Line chart showing demand curve with population adjustment	Line chart showing demand curve with population adjustment	Success
TC_07	Data Integration Validation	Merge groundwater, population, and demand CSVs	Clean merged dataset with no missing values	Clean merged dataset with no missing values	Success
TC_08	Input Data Validation	Check for missing/invalid values in CSVs	NaNs dropped, mismatches in district names resolved	NaNs dropped, mismatches in district names resolved	Success
TC_09	Multi-State Functionality	Validate Andhra Pradesh flow mirrors Telangana logic	Both states show groundwater & demand predictions	Both states show groundwater & demand predictions	Success

6. RESULTS

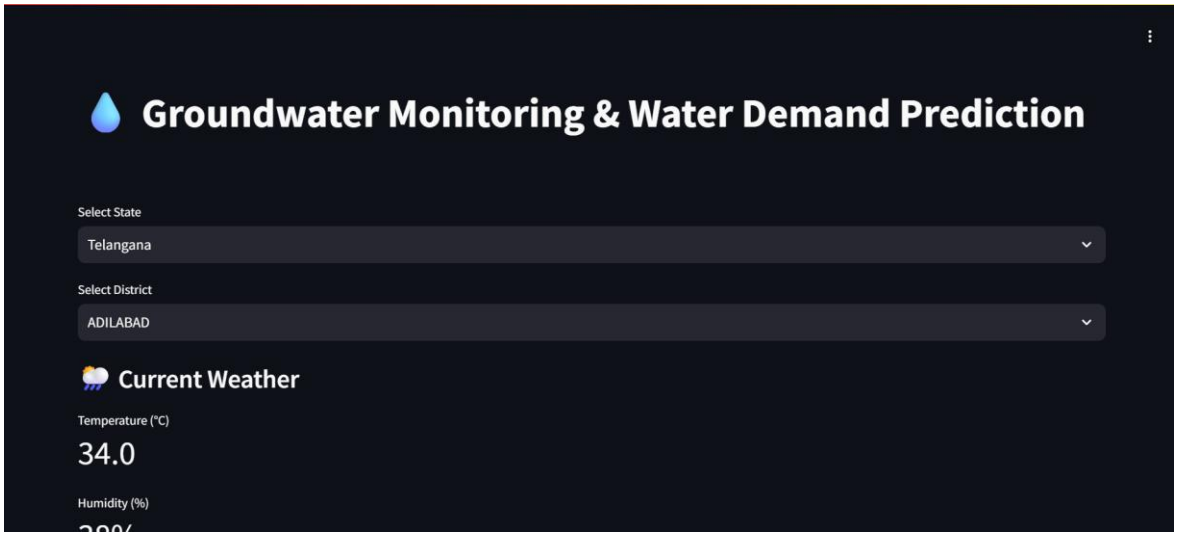


Fig. 6.1 Dropdown Selection

Fig. 6.1 shows the starting part of the Smart Water Analysis dashboard where the user selects the state and corresponding district of their choice.

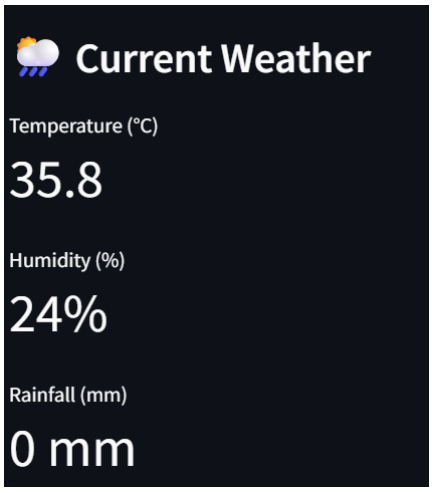


Fig. 6.2 Weather Data

Fig. 6.2 displays the weather data of the selected district and if data is not available then it does not show any weather data.

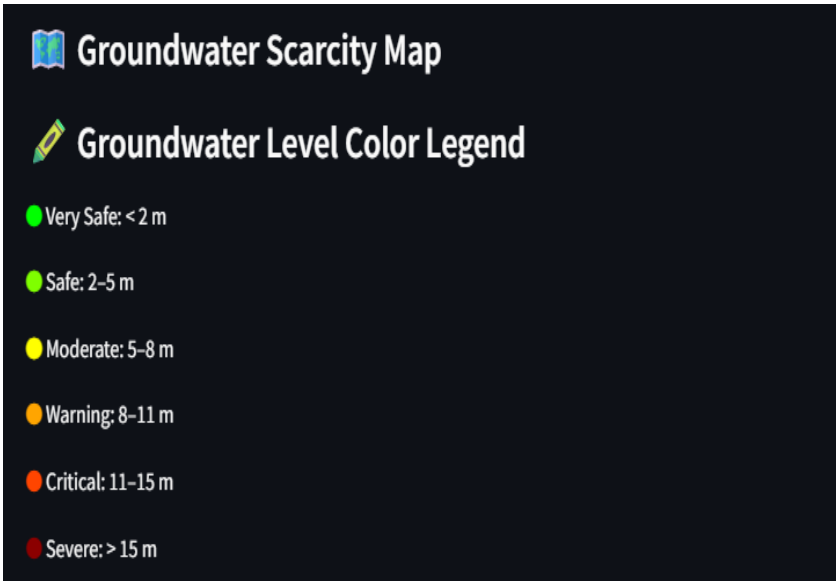


Fig. 6.3 Color Legend

Fig. 6.3 presents the color legend for the Groundwater Scarcity Choropleth Ma

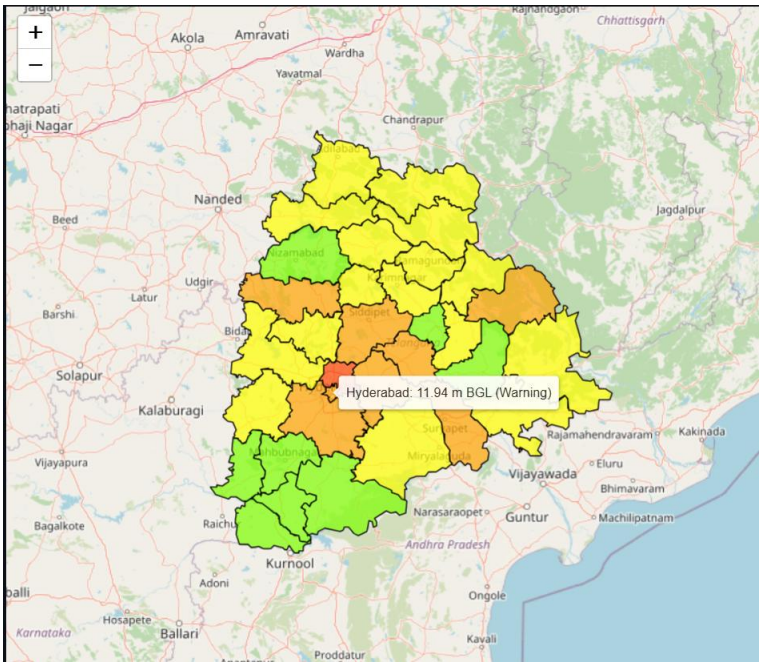


Fig. 6.4 Groundwater Scarcity Map

Fig. 6.4 displays the Groundwater Scarcity choropleth map with the color legend applied to it. Whenever the user hovers over any district, the district name along with the BGL and its Scarcity level are shown. Here we take the example of Telangana State.

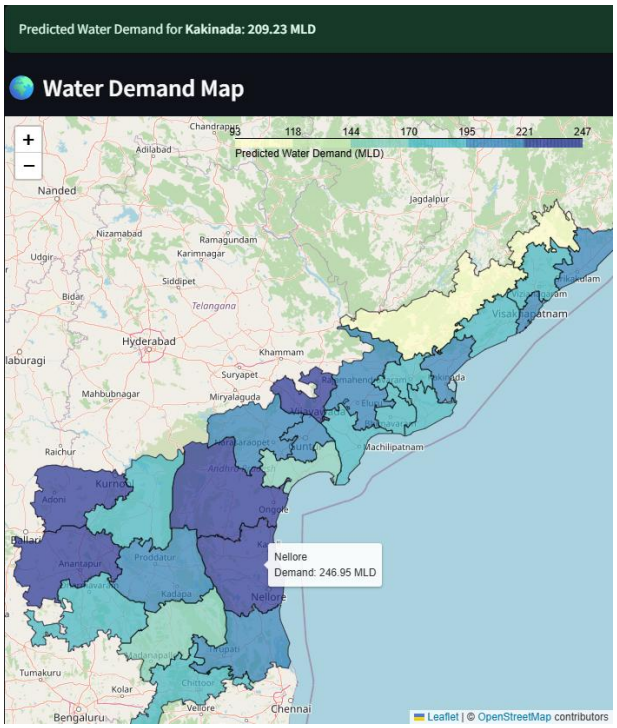


Fig 6.5 Estimated Water Demand Map

Fig. 6.5 displays the Estimated Water Demand Map of Andhra Pradesh and shows the numerical data for chosen district, here Kakinada. The color legend is displayed in map itself and successfully rendered as shown.

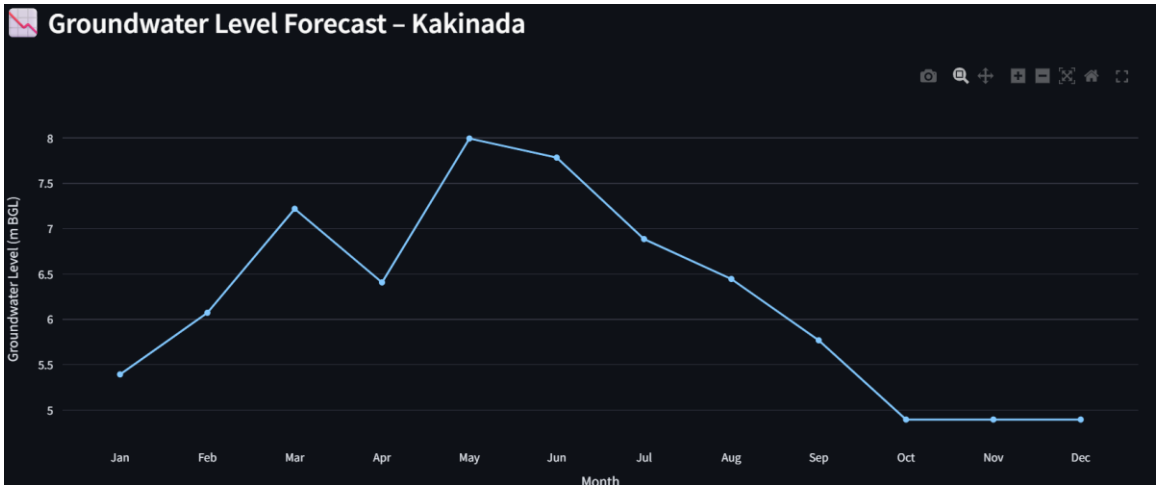


Fig 6.6 Groundwater Level Forecast

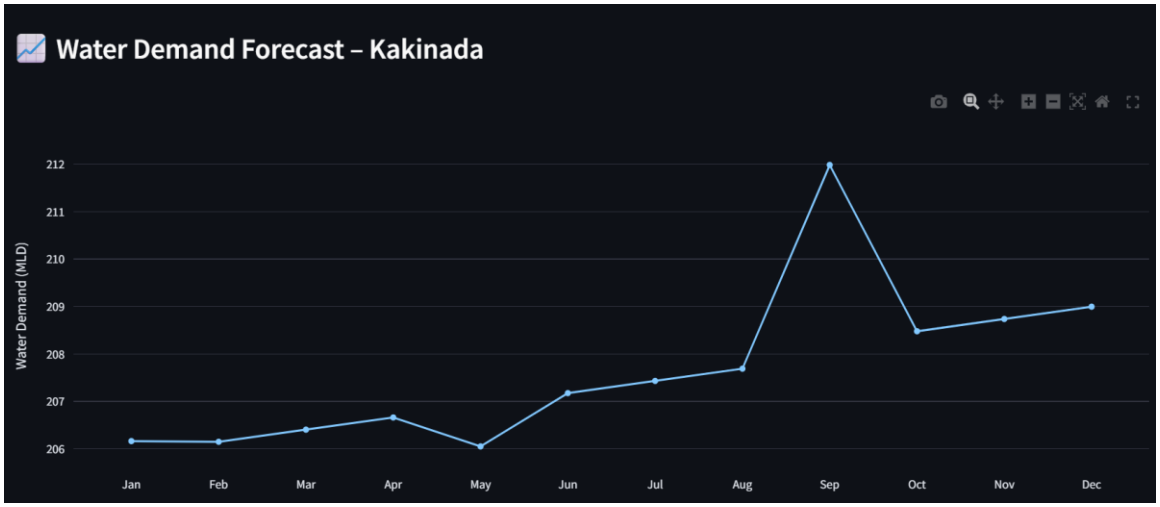


Fig 6.7 Water Demand Forecast

Fig 6.6 and 6.7 are both forecast line charts generated by model that predict the respective data for the next 12 months on how it is going to change.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The Smart Water Scarcity Analysis application provides a powerful, district-level visualization and prediction platform that integrates real-time environmental data, machine learning models, and geospatial mapping. It efficiently combines groundwater levels, population statistics, and rainfall data to produce insights on water scarcity and future demand. Users can interactively explore choropleth maps, track rainfall and temperature metrics, and monitor predicted water demand over time. By offering dynamic filtering by state and district, it ensures tailored information delivery. The use of Gradient Boosting Regressors allows reliable trend forecasting, enhancing policy-making and resource planning

Through consistent preprocessing, the data is sanitized for accuracy, and predictive models are trained per district, taking into account factors like soil recharge and climate conditions. The integration of future population growth rates ensures the water demand forecasts remain realistic and practical. The color-coded maps and legends provide a clear representation of the groundwater situation, helping users quickly identify critical regions.

Overall, the project successfully bridges environmental data science and usability. It lays a foundation for expanding to more states and deeper analytics.

7.2 FUTURE ENHANCEMENTS

Although the current version of Smart Water Scarcity Analysis is effective, several enhancements can improve its scalability, accuracy, and user engagement:

- **Advanced Forecasting Models**: Replace or supplement GBR with LSTM, XGBoost, or Prophet models for improved time-series prediction accuracy.
- **User Alerts and Notifications**: Schedule monthly reports for water management authorities.

- **Mobile-Friendly Interface**: Develop a mobile-responsive frontend or companion mobile app for on-field officers and farmers.
- **Dynamic Policy Simulation**: Allow users to simulate effects of policy interventions like artificial recharge, crop change, or industrial usage regulation on future water levels
- **Climate Change Scenario Modeling** : Incorporate IPCC RCP scenarios to predict long-term water stress under various climate trajectories.

8. REFERENCES

[1] CTian, Z.W. and Qian, R.L., 2024. Chinese Water Demand Forecast Based on iTransformer Model. IEEE Access.

[2] Arsene, D., Predescu, A., Stuparu, M., Truică, C.O., Mocanu, M. and Chiru, C., 2022, October. Predicting consumption events in a water monitoring system. In 2022 26th International Conference on System Theory, Control and Computing (ICSTCC) (pp. 74-79). IEEE.

[3] Elmotawakkil, Abdessamad, Abdelkhalik Sadiki, and Nourddine Enneya. "Predicting groundwater level based on remote sensing and machine learning: a case study in the Rabat-Kénitra region." Journal of Hydroinformatics 26.10 (2024): 2639-2667.

[4] Allen, M., Preis, A., Iqbal, M. and Whittle, A.J., 2013, July. Water distribution system monitoring and decision support using a wireless sensor network. In 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (pp. 641-646). IEEE.

[5] Jain, S., Parida, A.K. and Sankaranarayanan, S., 2020. Water scarcity prediction for global region using machine learning. International Journal of Water, 14(1), pp.69-88.

[6] Li, W., Finsa, M.M., Laskey, K.B., Houser, P. and Douglas-Bate, R., 2023. Groundwater level prediction with machine learning to support sustainable irrigation in water scarcity regions. Water, 15(19), p.3473.

[7] National Remote Sensing Centre (NRSC), Bhoonidhi – Indian EO Data Discovery and Archive Platform , 2025, https://bhoonidhi.nrsc.gov.in/bhoonidhi/home.html.