Rising Waters: A Machine Learning Approach to Flood Prediction

CHAPTER-1

INTRODUCTION

Floods are one of the most dangerous natural disasters, causing serious damage to life, property, and the environment. Due to climate change and unpredictable weather, floods are becoming more frequent and harder to predict. Traditional methods are often slow and not always accurate.

The project **"Rising Waters: A Machine Learning Approach to Flood Prediction"** uses machine learning to analyse weather and water-level data to predict floods early. This helps authorities and people take timely action and reduce damage caused by floods.

1.1 project overview:

Floods are among the most devastating natural disasters, causing severe loss of life, property, and economic stability every year. With climate change increasing the frequency and intensity of extreme rainfall events, traditional flood prediction methods are no longer sufficient to provide timely and accurate warnings. This project, **"Rising Waters: A Machine Learning Approach to Flood Prediction,"** aims to develop an intelligent system that can predict flood risks using historical and real-time environmental data.

The system uses **machine learning algorithms** to analyse multiple factors such as rainfall, river water levels, temperature, soil moisture, and past flood records. By learning patterns

from this data, the model can identify early warning signs of flooding and estimate the likelihood of a flood occurring in a given area.

The project focuses on building a **data-driven predictive model** that is more accurate and faster than traditional statistical methods. The model is trained on historical flood and weather datasets and tested on unseen data to ensure reliability. Once trained, the system can provide **real-time flood risk predictions**, helping authorities and communities take preventive action.

1.2 Purpose and Objectives of the project:

The purpose of the project is to develop an intelligent and reliable system that can predict floods in advance using machine learning techniques. The system aims to analyse environmental and weather-related data to provide early warnings, helping reduce the loss of life, property, and economic damage caused by floods.

The main objectives of this project are:

- To collect and analyze historical and real-time data related to rainfall, river water levels, temperature, and previous flood events.
- To build a machine learning model that can accurately predict the possibility of flooding.
- To identify patterns and trends in environmental data that lead to flood situations.
- To provide early warnings for flood-prone areas to support timely decision-making.

Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

2.1 PROBLEM STATEMENT:

Floods are among the most destructive natural disasters, causing significant loss of life, property damage, economic disruption, and environmental degradation worldwide. With the increasing impacts of climate change, rapid urbanization, and deforestation, the frequency and intensity of flooding events have risen substantially. Traditional flood forecasting systems often rely on manual monitoring, static hydrological models, or delayed reporting mechanisms, which may not provide accurate, real-time, and location-specific predictions.

There is a critical need for an intelligent, data-driven system capable of analysing large volumes of environmental and meteorological data to predict flood occurrences with high accuracy and timeliness. Factors such as rainfall intensity, river water levels, soil moisture, temperature, drainage capacity, topography, and historical flood records contribute to flood risk. However, integrating and interpreting these diverse datasets efficiently remains a major challenge.

The problem this project aims to address is the development of a robust Machine Learning-based flood prediction model that can:

- Analyse historical and real-time environmental data
- Identify patterns and early warning indicators of flooding
- Predict the likelihood and severity of flood events
- Provide timely alerts to reduce risk and improve disaster preparedness

By leveraging advanced machine learning algorithms, data analytics, and predictive modelling techniques, the proposed system seeks to enhance early warning capabilities, support decision-makers, and ultimately minimize the socio-economic impact of floods.

Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	4 MARKS

2.2 EMPATHY MAP CANVAS

EMPATHY MAP:

THINK & FEEL

- Worried about sudden and unpredictable flood events
- Fear of casualties and infrastructure damage
- Pressure to ensure public safety
- Concerned about accuracy of existing forecasting systems
- Motivated to improve preparedness and response time

SEE

- Increasing frequency of floods due to climate change
- Damage to roads, homes, and public infrastructure
- Communities living in high-risk zones
- Limited integration of real-time data systems

HEAR

- Climate experts warning about extreme weather risks
- Public demanding better early warning systems
- Government policies on disaster preparedness
- Media criticism during disaster mismanagement

SAY & DO

- “We need accurate and real-time flood predictions.”
- Monitor rainfall and river-level dashboards
- Plan evacuation strategies
- Allocate emergency funds and resources
- Coordinate with response teams

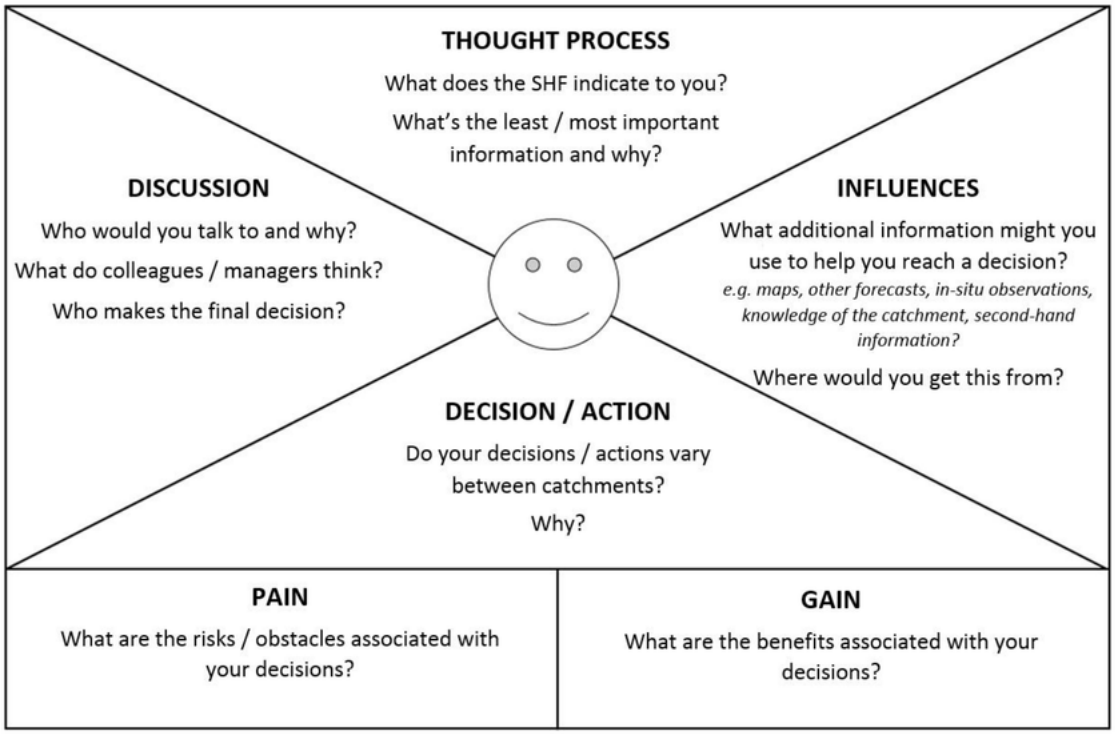
PAINS

- Delayed or inaccurate flood warnings
- Data silos between departments

- High economic losses
- Slow emergency response
- Public distrust after disasters

GAINS

- Reliable real-time prediction system
- Faster evacuation planning
- Reduced loss of life and property
- Data-driven decision making
- Increased public trust and resilience



Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	4 MARKS

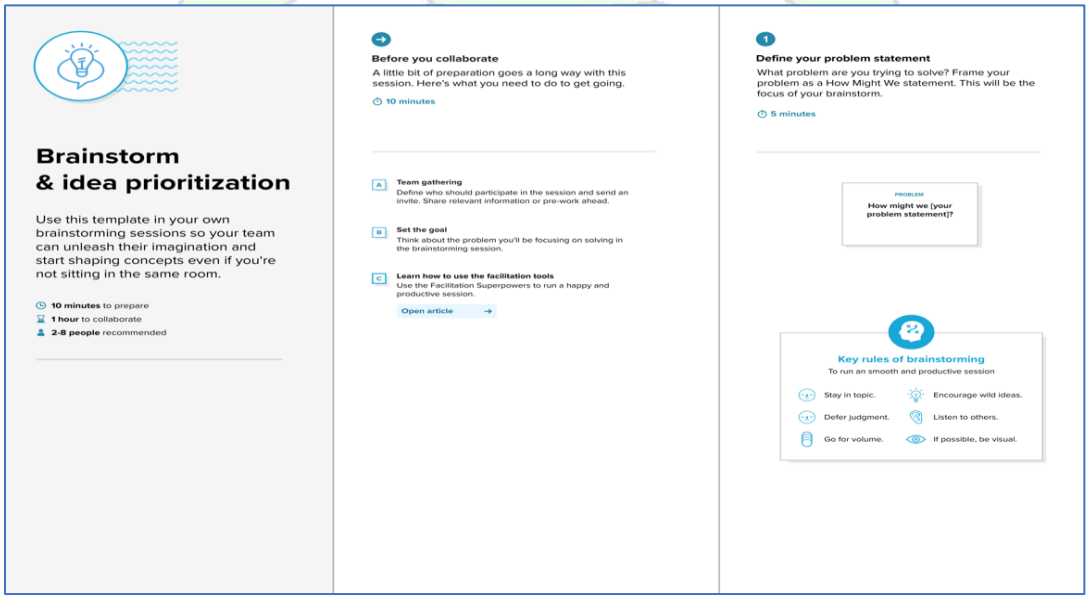
2.3 BRAINSTORING:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

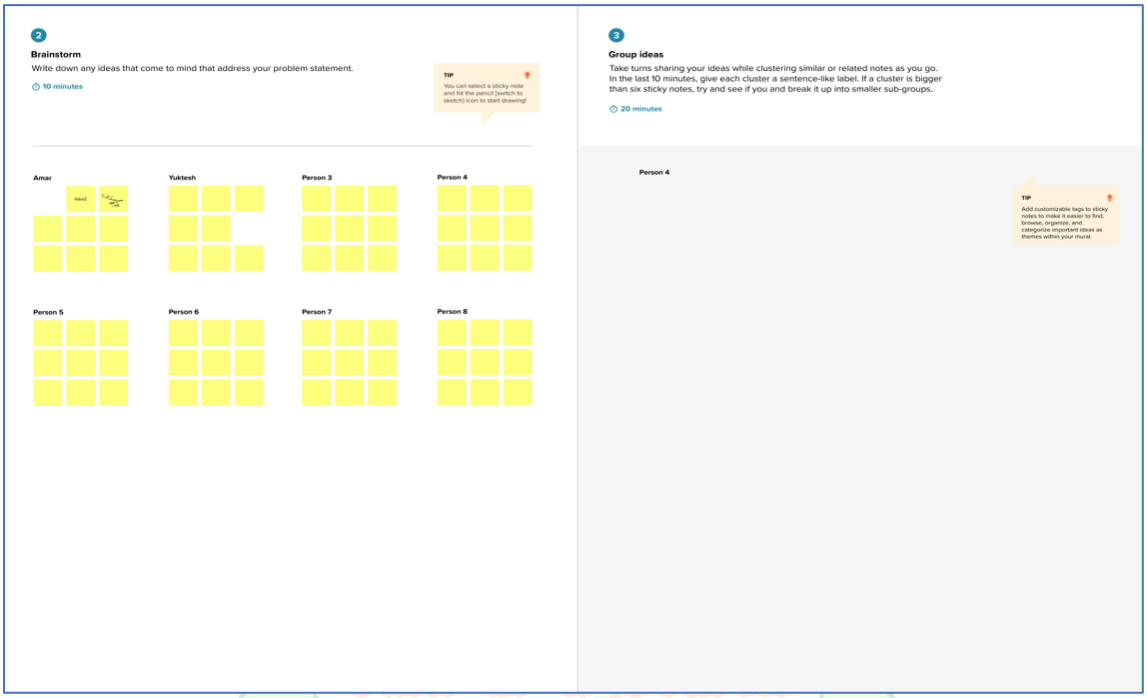
Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Reference: https://www.mural.co/templates/brainstorm-and-idea-prioritization

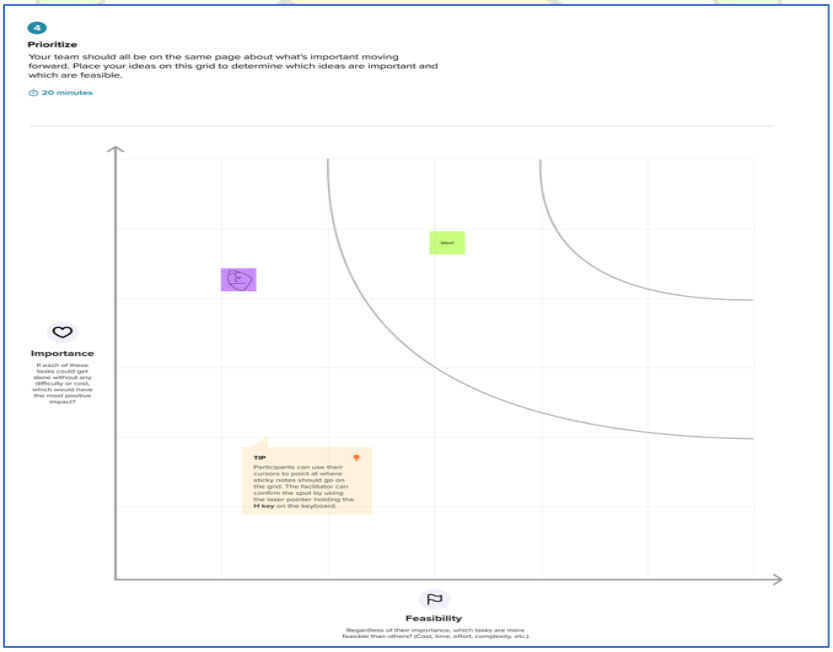
Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping



Step-3: Idea Prioritization



DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

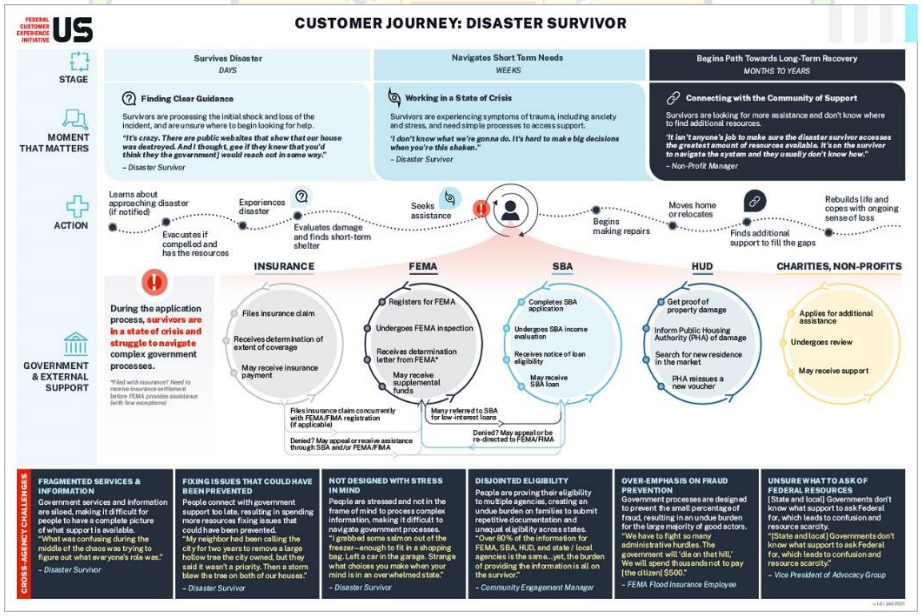
3.1 CUSTOMER JOURNEY MAP:

The Customer Journey Map illustrates the steps a disaster management authority or local government official takes while using the Rising Waters machine learning-based flood prediction system. It visualizes the journey from awareness of flood risks to the outcome of successful flood management.

Each stage highlights:

- **User actions** – what the authority does at that stage
- **Touchpoints** – dashboards, alerts, and reports
- **Emotions** – how the user feels, from concern to confidence
- **Opportunities** – how the system supports faster, data-driven decision-making

By mapping this journey, the diagram shows how *Rising Waters* helps authorities anticipate floods, plan evacuations, and reduce casualties and damage, making the system user-focused and impactful.



DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

3.2 SOLUTION REQUIREMENT:

Functional Requirements:

These define what the system should do:

1. **Real-Time Flood Prediction:** Use machine learning algorithms to predict flood events based on historical and live data.
2. **Data Integration:** Collect and process data from rainfall sensors, river gauges, weather APIs, and satellite imagery.
3. **Early Warning Alerts:** Generate automated alerts via SMS, email, or dashboard notifications.
4. **Visualization Dashboard:** Display flood risk heatmaps, water levels, and probability charts for easy decision-making.
5. **User Management:** Allow different access levels for government officials, disaster teams, and urban planners.
6. **Scenario Simulation:** Provide "what-if" scenarios for potential flood events and mitigation planning.

Non-Functional Requirements:

These define how the system should perform:

1. **Accuracy:** Predictions should achieve a high degree of reliability (target ≥ 85% accuracy).
2. **Scalability:** Handle large datasets from multiple regions simultaneously.
3. **Performance:** Provide real-time updates with minimal latency (<5 minutes).

4. **Usability:** Intuitive interface for non-technical users with clear visualization.
5. **Security:** Secure sensitive data and user credentials with authentication.
6. **Availability:** 24/7 system uptime, especially during flood season.

Technical Requirements

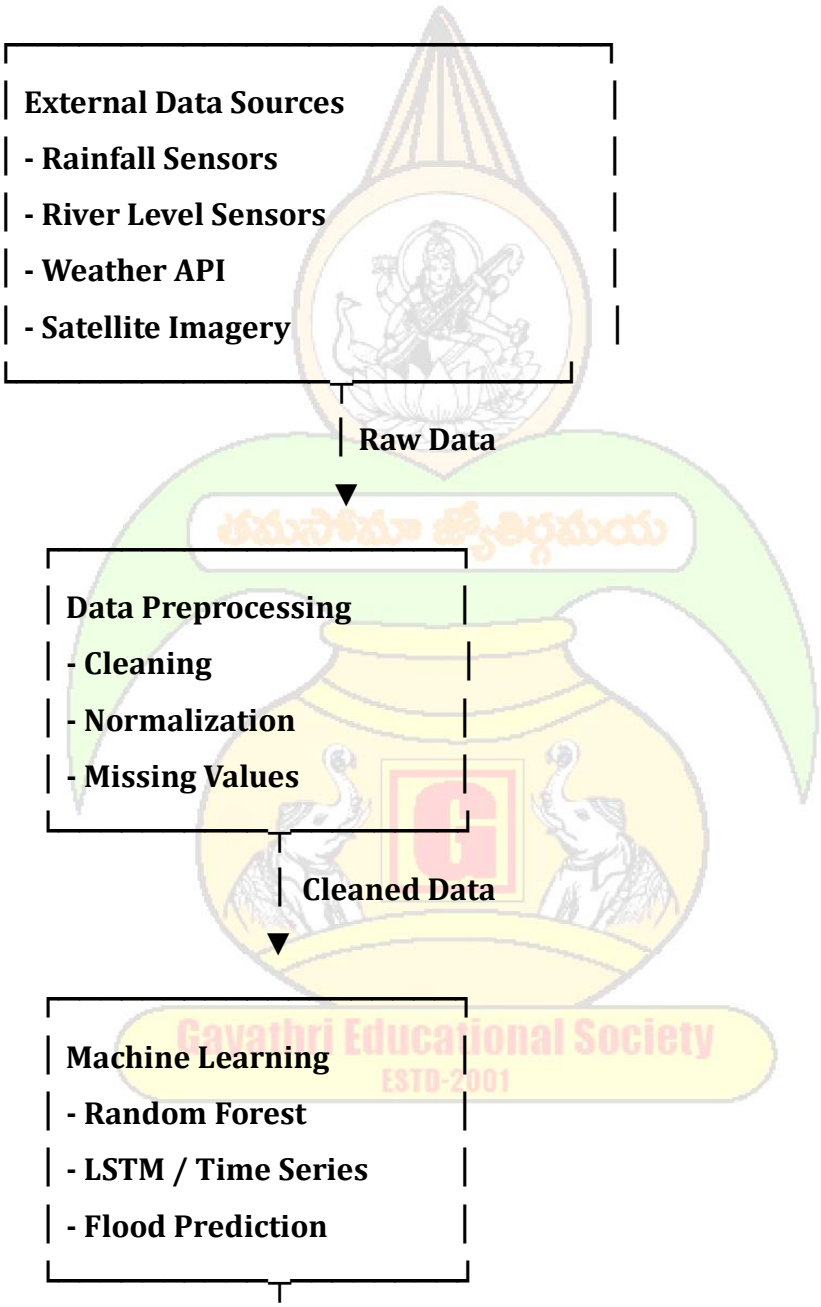
- **Programming Languages:** Python (for ML), JavaScript/React (for dashboard)
- **Machine Learning Models:** Random Forest, LSTM, or Gradient Boosting for flood prediction
- **Database:** SQL/NoSQL for sensor and historical data storage
- **APIs:** Weather APIs, IoT river sensor integration
- **Cloud Hosting:** AWS, Azure, or Google Cloud for real-time data processing

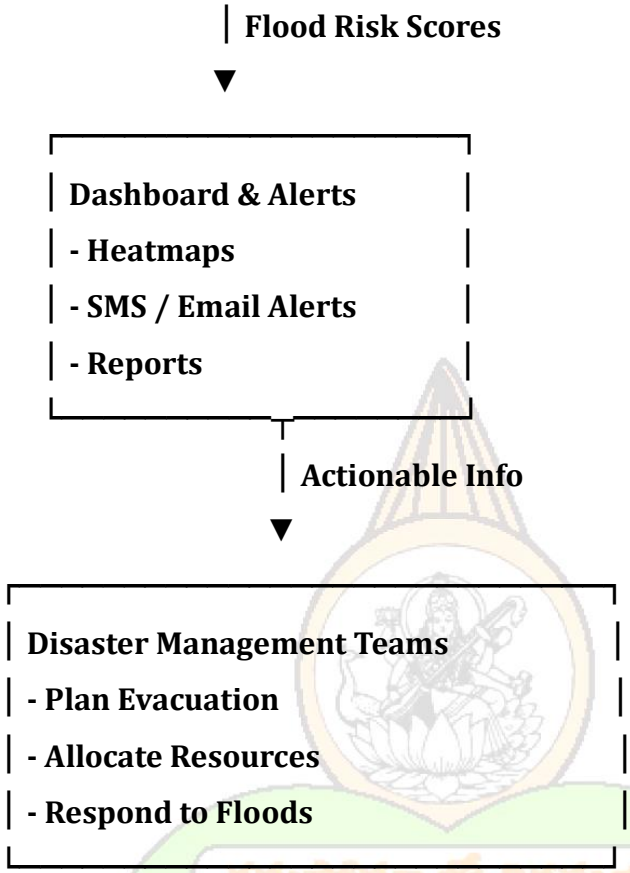
Business / User Requirements:

- Enable authorities to make faster, informed decisions
- Reduce loss of life and property during floods
- Provide a cost-effective early warning system
- Enhance public trust through transparent flood information

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	3 MARKS

3.3 DATA FLOW DIAGRAM:





Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	3 MARKS

3.4 TECHNOLOGY STACK:

1. Data Acquisition

Flood prediction relies on diverse datasets. Common technologies and sources include:

- **Remote Sensing & Satellite Data**
  - Platforms: NASA EarthData, Copernicus Sentinel Hub
  - Libraries: sentinelsat, rasterio, GDAL
- **Meteorological Data**
  - Sources: NOAA, Open WeatherMap API, Weather Underground
  - Tools: Python requests, pandas
- **Hydrological Data**
  - River flow, rainfall, water levels from local or national agencies
  - APIs or CSV/NetCDF datasets

2. Data Storage & Management:

Handling large-scale geospatial and temporal data requires:

- **Databases**
  - PostgreSQL + PostGIS – for geospatial data
  - MongoDB – for semi-structured or JSON data
- **Cloud Storage**
  - AWS S3, Google Cloud Storage, Azure Blob Storage
- **Data Formats**
  - CSV, NetCDF, GeoTIFF, HDF5

3. Data Preprocessing & Feature Engineering:

Preparing data for ML models involves:

- **Python Libraries**
  - pandas, numpy – data wrangling
  - scikit-learn – scaling, encoding, and preprocessing
  - xarray, rasterio, geopandas – for geospatial/temporal analysis

  - matplotlib, seaborn, plotly – visualization of flood patterns
- **Techniques**
  - Handling missing data
  - Normalization/standardization
  - Feature extraction (e.g., rainfall accumulation, river level change, slope, soil saturation)

4. Machine Learning & Modeling

Choosing predictive models for flood forecasting:

- **Algorithms**
  - **Classical ML**: Random Forest, Gradient Boosting (XGBoost, LightGBM), Support Vector Machines
  - **Deep Learning**: LSTM, GRU (for time-series prediction)
  - **Hybrid Models**: CNN-LSTM for spatio-temporal patterns
- **Frameworks**
  - scikit-learn
  - TensorFlow / Keras
  - PyTorch
  - XGBoost / LightGBM
- **Model Evaluation**
  - Metrics: RMSE, MAE, Precision/Recall (if classification), ROC-AUC
  - Cross-validation and time-series split

5. Deployment & Visualization:

Once the model is trained, deployment and visualization are key:

- **Web & Dashboard**
  - Streamlit or Dash for interactive flood dashboards
  - Flask or FastAPI for backend APIs
- **GIS & Mapping**
  - Leaflet.js, Folium, Kepler.gl – for real-time flood maps
- **Cloud & CI/CD**
  - AWS EC2, GCP Compute Engine, Azure ML
  - Containerization with Docker, orchestration with Kubernetes
- **Monitoring**
  - Prometheus + Grafana for monitoring model predictions and data pipelines

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	5 MARKS

4.1 PROBLEM SOLUTION FIT

1. The Problem:

Flooding is one of the most destructive natural disasters worldwide, affecting millions of people every year. Key challenges include:

- Unpredictable weather patterns due to climate change
- Delayed or inaccurate flood warnings
- Limited use of real-time data
- Inadequate early response systems
- High economic losses and infrastructure damage

According to global disaster monitoring agencies, floods account for a significant percentage of natural disaster-related damages annually. Many regions, especially developing areas, lack accurate predictive systems that combine historical and real-time environmental data.

Core Pain Points:

- Communities receive warnings too late
- Emergency services lack predictive planning tools
- Governments rely on static threshold-based systems
- Traditional hydrological models struggle with nonlinear patterns

2. Target Users

- Government disaster management authorities
- Environmental agencies
- Urban planners
- Insurance companies
- NGOs working in climate resilience
- Local communities in flood-prone regions

3. Current Solutions & Gaps

Traditional Methods:

- Rule-based flood threshold systems
- Manual rainfall and river-level monitoring
- Satellite-only analysis
- Static hydrological simulation models

Gaps:

- Poor scalability
- Limited predictive accuracy
- Lack of machine learning integration
- Inability to process large multi-source datasets

4. How the Solution Fits the Problem

Problem	Solution Fit
Late warnings	Predictive modeling provides early alerts
Nonlinear weather patterns	ML captures complex relationships
Data fragmentation	Centralized multi-source data pipeline
Static models	Adaptive learning improves over time
Lack of actionable insights	Risk scoring + dashboard visualization

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	5 MARKS

4.2 PROPOSED SOLUTION

Objective

To develop an intelligent **Machine Learning-based flood prediction system** that analyses historical and real-time environmental data to predict the likelihood of floods and provide early warnings to reduce damage and loss of life.

Solution Overview

The proposed system will:

- Collect historical weather and river data
- Preprocess and clean the dataset
- Train multiple machine learning models
- Select the best-performing model
- Deploy the model using a web application
- Provide flood risk prediction (Low / Medium / High)

Data Sources

The model will use reliable data from:

- India Meteorological Department
- Central Water Commission
- Public datasets (e.g., Kaggle flood datasets)

Alert Mechanism

Future enhancement:

- SMS alerts
- Real-time IoT sensor integration
- Government disaster management integration

Expected Outcome

- Early flood prediction
- Reduced damage and casualties

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	5 MARKS

4.3 SOLUTION ARCHITECTURE:

A. Data Collection Layer

This layer gathers historical and real-time environmental data from trusted sources such as:

- India Meteorological Department
- Central Water Commission
- Public datasets (e.g., Kaggle)

Collected Parameters:

- Rainfall (mm)
- Temperature (°C)
- Humidity (%)
- Wind speed
- Historical flood records

B. Data Processing Layer

This layer prepares raw data for machine learning.

Key Processes:

- Data cleaning (handling missing values)
- Removing outliers
- Feature scaling/normalization
- Encoding categorical values
- Splitting dataset into training and testing sets

This improves model reliability and performance.

C. Machine Learning Layer

This is the core intelligence of the system.

Algorithms Used:

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine
- XGBoost

D. Deployment Layer

The trained model is deployed as a web application using:

- **Programming Language:** Python
- **Framework:** Flask
- **Frontend:** HTML, CSS

The model is loaded into the Flask application and accepts user inputs for prediction.

E. User Interface Layer

Users interact with the system through a simple web interface.

The interface allows users to:

- Enter environmental parameters
- Click Predict
- View flood risk level

F. Alert & Notification Layer

If the predicted probability exceeds a predefined threshold:

- A warning message is displayed
- Risk level is highlighted
- (Future Scope) SMS alerts and integration with disaster management authorities

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	1 MARKS

CHAPTER-5

Project Planning & Scheduling

5.1 – PROJECT MILESTONES AND TASKS

1 Data Collection

Objective: Gather high-quality, relevant data required for the project.

Tasks:

- Identify reliable data sources (databases, APIs, web scraping, surveys, etc.)
- Collect structured and unstructured data
- Ensure data relevance and completeness
- Store data in a centralized database or storage system
- Maintain data documentation for reference

Deliverables:

- Raw dataset
- Data source documentation
- Data storage setup

2 Data Pre-Processing

Objective: Clean and prepare the data for model training and analysis.

Tasks:

- Remove duplicates and handle missing values
- Handle outliers and inconsistent data
- Data normalization or scaling
- Feature engineering and selection
- Encode categorical variables
- Split dataset into training and testing sets

Deliverables:

- Cleaned dataset

- Feature-engineered dataset
- Preprocessing scripts

3 Model Building

Objective: Develop and train a machine learning model.

Tasks:

- Select appropriate algorithm(s)
- Train model using training dataset
- Hyperparameter tuning
- Model validation and evaluation
- Compare performance metrics
- Finalize best-performing model

Deliverables:

- Trained model
- Evaluation report (Accuracy, Precision, Recall, F1-score, etc.)
- Saved model file

4 API Integration

Objective: Expose the trained model via an API for external access.

Tasks:

- Develop REST API endpoints
- Integrate model with backend framework (Flask / FastAPI / Django)
- Implement request validation
- Add error handling
- Test API responses
- Secure API (authentication if required)

Deliverables:

- Functional API
- API documentation (Swagger/Postman collection)
- Deployment-ready backend

5 Web Integration

Objective: Integrate API into a web-based user interface.

Tasks:

- Design user-friendly frontend interface
- Connect frontend to backend API
- Display model predictions dynamically
- Implement input validation
- Perform end-to-end testing
- Deploy web application

Deliverables:

- Fully functional web application
- User interface design
- Live deployment link (if applicable)

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

5.2 SPRINT DELIVERY PLAN:

🔷 Phase 1: Live Sessions (Week 1–6)

Objective: Build strong foundational knowledge and prepare interns for real-time project development.

Sprint 1 (Week 1–2): Fundamentals & Tools

Focus Areas:

- Introduction to Internship Program
- Programming Fundamentals (Python / Relevant Tech Stack)
- Git & GitHub
- Development Environment Setup
- Basics of Databases

Deliverables:

- Setup development environment
- GitHub repository creation
- Mini practice assignments

Sprint 2 (Week 3–4): Data & Backend Foundations

Focus Areas:

- Data Handling (Pandas / Data Structures)
- Data Cleaning Techniques
- Introduction to APIs
- Backend Basics (Flask / FastAPI)
- SQL & Database Integration

Deliverables:

- Data preprocessing assignment
- Basic API development task

- Database connectivity demo

Sprint 3 (Week 5–6): Machine Learning & Deployment Basics

Focus Areas:

- Machine Learning Fundamentals
- Model Training & Evaluation
- REST API Integration with Model
- Introduction to Web Integration
- Deployment Overview

Deliverables:

- Simple ML Model
- Model evaluation report
- API with working prediction endpoint

◆ **Phase 2: Project Work (Week 7–15)**

Objective: Apply learned skills to build a complete end-to-end project.

Sprint 4 (Week 7–8): Project Planning & Data Collection

Activities:

- Finalize project topic
- Define problem statement
- Collect dataset
- Perform initial data analysis (EDA)

Deliverables:

- Project proposal document
- Dataset documentation
- EDA report

Sprint 5 (Week 9–10): Data Preprocessing & Feature Engineering

Activities:

- Clean dataset
- Handle missing values & outliers
- Feature engineering

- Data transformation
- Train-test split

Deliverables:

- Cleaned dataset
- Preprocessing pipeline
- Feature documentation

Sprint 6 (Week 11–12): Model Development & Optimization

Activities:

- Train multiple models
- Hyperparameter tuning
- Model comparison
- Performance evaluation

Deliverables:

- Best performing model
- Evaluation metrics report
- Saved model artifact

Sprint 7 (Week 13–14): API Development & Integration

Activities:

- Develop REST API
- Integrate trained model
- Implement validation & error handling
- Test API endpoints

Deliverables:

- Functional API
- API documentation
- Backend deployment

Sprint 8 (Week 15): Web Integration & Final Deployment

Activities:

- Develop frontend interface

- Connect frontend with API
- End-to-end testing
- Deployment & final presentation

Deliverables:

- Fully functional web application
- Deployment link
- Final project presentation
- Internship completion report

Final Outcome

By the end of 15 weeks, interns will have:

- Strong technical foundation
- Real-time project experience
- A complete end-to-end deployed project
- Industry-ready portfolio project

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	1 MARKS

5.3 – PROJECT PROGRESS TRACKING:

1. Zoho Cliq Workspace Structure

◈ Channels Setup

Create the following channels for organized communication:

📌 1.announcements

- Official updates
- Sprint start/end notifications
- Deadlines & evaluation updates
- Meeting schedules

📌 2.project-discussion

- Technical queries
- Implementation discussions
- Code review discussions
- Issue troubleshooting

📌 3.daily-updates

- Daily progress reports
- Blockers
- Completed tasks

📌 4.resources

- Shared datasets
- Documentation links
- Recorded session links
- API documentation

📌 5.team-specific-channels (if multiple teams)

Example:

- #team-alpha
- #team-beta

2. Sprint-Based Tracking Method

Each sprint will follow a structured reporting cycle.

Daily Progress Update Format (Posted in #daily-updates)

Every intern must post:

Format:

Date:

Sprint:

Tasks Completed:

Tasks In Progress:

Blockers (if any):

Plan for Tomorrow:

Weekly Sprint Review (Every Weekend)

Mentor will post:

- Sprint Goals
- Completed Milestones
- Pending Tasks
- Risk Areas
- Next Week Targets

3. Task Tracking System

Option 1: Zoho Cliq Tasks Feature

Use built-in task management in Zoho Cliq:

- Assign tasks to interns

- Set deadlines
- Track completion status
- Add task priority (High / Medium / Low)

Task Status Workflow:

- **□ To Do**
- **🔘 In Progress**
- **□ Completed**
- **⚫ Blocked**

Option 2: Zoho Projects Integration (Optional)

For advanced tracking, integrate with:

- Zoho Projects

Use:

- Kanban Board
- Gantt Chart
- Milestone tracking
- Automated reminders

4. Sprint Progress Monitoring Dashboard

Track the following metrics weekly:

- % Tasks Completed
- API Development Progress
- Model Accuracy Improvement
- Deployment Readiness
- Bug Count
- Attendance in Live Sessions

Mentor shares a weekly progress summary in:
👉 announcements channel

5. Milestone Tracking Structure

Milestone	Week	Status	Owner	Remarks
Data Collection	Week 7–8	☐	Team	
Data Preprocessing	Week 9–10	☐	Team	
Model Building	Week 11–12	☐	Team	
API Integration	Week 13–14	☐	Team	
Web Integration	Week 15	☐	Team	

6. Escalation Process

If blocker > 24 hours:

- **Post in #project-discussion**
- **Tag mentor**
- **If unresolved → Schedule quick call via Zoho Cliq**
- **Update resolution summary in channel**

7. Performance Evaluation Criteria

Evaluation will be based on:

- Daily update consistency
- Sprint milestone completion
- Code quality
- Participation in discussions
- Final project delivery
- Timely submissions

Weekly Workflow Summary

Day	Activity
Monday	Sprint planning post
Tue–Thu	Development & daily updates

Friday	**Progress review**
Saturday	**Sprint demo**
Sunday	**Feedback & planning**

✅ **Final Outcome**

Using Zoho Cliq ensures:

- Clear communication
- Accountability
- Structured sprint tracking
- Real-time problem resolution
- Professional workflow management

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	1 MARKS

5.4 - TEAM MANAGEMENT TOOLS FOR AGILE PLANNING:

What is Jira?

Jira is an Agile project management and issue-tracking tool developed by Atlassian. It supports Scrum and Kanban methodologies, helping teams plan, track, and release software efficiently.

Jira Project Structure for Internship

1 Project Creation

Create a project with:

- Project Name: Internship Capstone Project
- Template: Scrum (Recommended)
- Project Type: Software Development

Issue Types Configuration

Define standard issue types:

- Epic – Major project phases
- Story – Feature or functionality
- Task – Smaller implementation steps
- Bug – Errors or defects
- Sub-task – Breakdown of tasks

Suggested Epics (Based on Your Milestones)

Epic	Description
Data Collection	**Dataset gathering & validation**

Data Preprocessing	**Cleaning & feature engineering**
Model Development	**ML training & evaluation**
API Integration	**Backend & model API**
Web Integration	**Frontend & deployment**

Sprint Planning Structure

◆ Sprint Duration

- **2 Weeks per Sprint**
- **Total: 4–5 Sprints (Project Phase)**

◆ Sprint Workflow

1. **Create Sprint in Backlog**
2. **Add Stories/Tasks to Sprint**
3. **Assign Tasks to Interns**
4. **Set Priority (High/Medium/Low)**
5. **Estimate Story Points**
6. **Start Sprint**

Workflow Status Configuration

Customize workflow:

- To Do
- In Progress
- In Review
- Done
- ● Blocked

This ensures transparent tracking of task movement.

Agile Boards in Jira

1 Scrum Board

Used for:

- Sprint planning
- Daily standups
- Tracking sprint progress

2 Kanban Board (Optional)

Used for:

- Continuous workflow
- API & bug tracking

📈 Agile Reports for Monitoring

Jira provides built-in reports:

- 📉 Burndown Chart – Sprint progress tracking
- 📈 Velocity Chart – Team performance
- 📋 Sprint Report – Completed vs pending work
- 🐞 Bug Report – Defect tracking

Mentors can review reports weekly to evaluate progress.

👥 Role-Based Access

Define permissions:

Role	Responsibilities
Project Admin	Configure board & workflow
Scrum Master	Sprint planning & review
Developer (Intern)	Task implementation
Reviewer	Code & feature review

Daily Standup Format (Using Jira Board)

Each intern updates:

- What was completed yesterday
- What will be done today
- Any blockers

Tasks must be moved across workflow stages accordingly.

Integration Capabilities

Jira can integrate with:

- GitHub (code tracking)
- CI/CD tools
- Slack or Zoho Cliq (notifications)
- Confluence (documentation)

Example Sprint Breakdown (2 Weeks)

Sprint Goal: Complete Data Preprocessing

Planned Stories:

- Clean missing values
- Outlier detection
- Feature scaling
- Train-test split

Expected Clean dataset ready for model training.

Benefits of Using Jira

- Structured Agile Planning
- Clear Accountability
- Transparent Sprint Tracking
- Performance Analytics
- Real-time Status Visibility
- Industry-standard project management tool

📌 **Final Outcome**

By implementing Jira for Agile planning, the internship project will follow:

- Scrum-based sprint execution
- Organized backlog management
- Efficient task delegation
- Measurable progress tracking
- Professional software development workflow

Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

CHAPTER-6

6.1 PRE-REQUISITES:

To complete this project, we require the following software's, and packages.

- Anaconda Navigator
- PyCharm
- Python

ANAKONDA NAVIGATER:

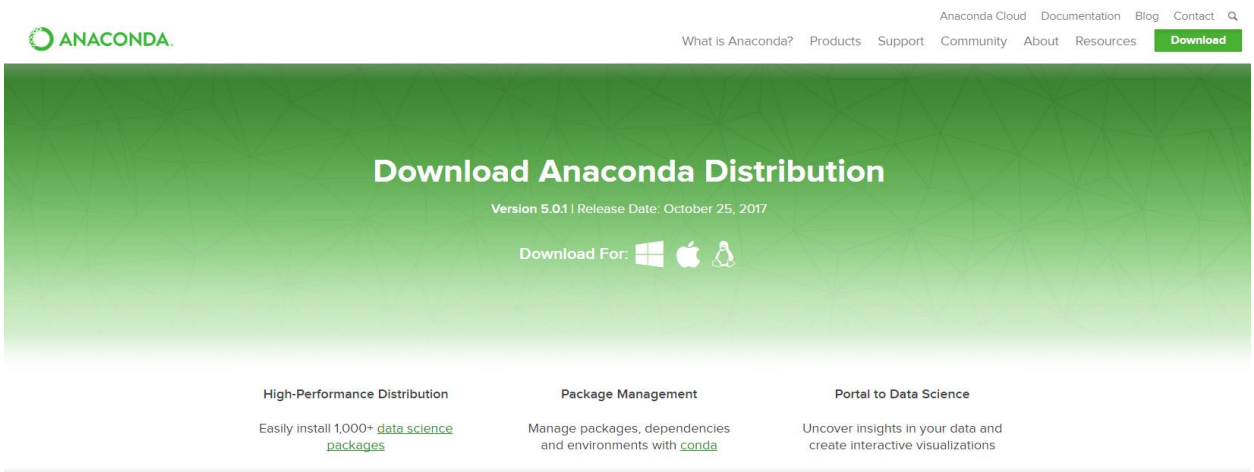
How to Install Anaconda on Windows:

Anaconda is a popular open-source distribution of Python and R and is widely used in the field of data science, machine learning and scientific computing. It contains Jupyter, Sypder, etc. that are well capable of handling a large number of data sets and processes as per user's need. It helps in simplifying package management and deployment, which promotes a robust environment for various data-related tasks.

Step 1: Visit the Official Website:

Download Anaconda from https://www.anaconda.com .

Make sure to download the "Python 3.13.1 Version".



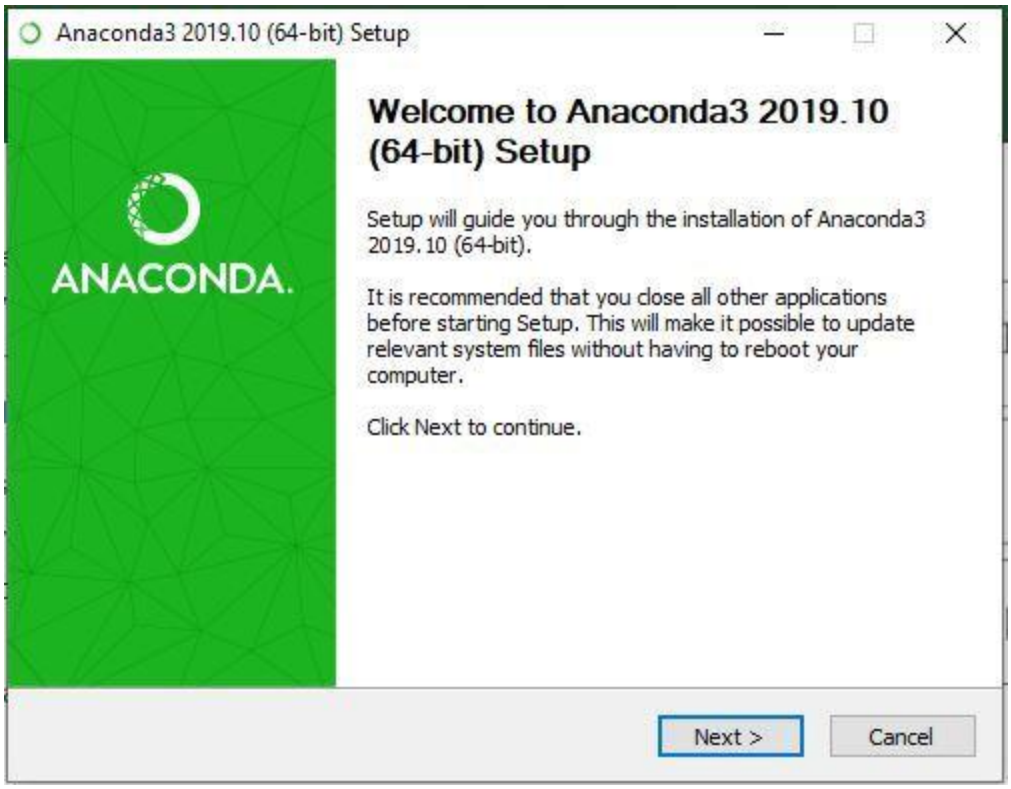
Step 2: Select the Windows Installer.

- Click on the Download button.

- Under the operating system options, select Windows.
- Choose Windows Installer (64-bit) .
- Download the .exe installer file.
- Select the location where you want to save the file and click **"Save"** to start the downloading process.

Step3: Run the Anaconda Installer

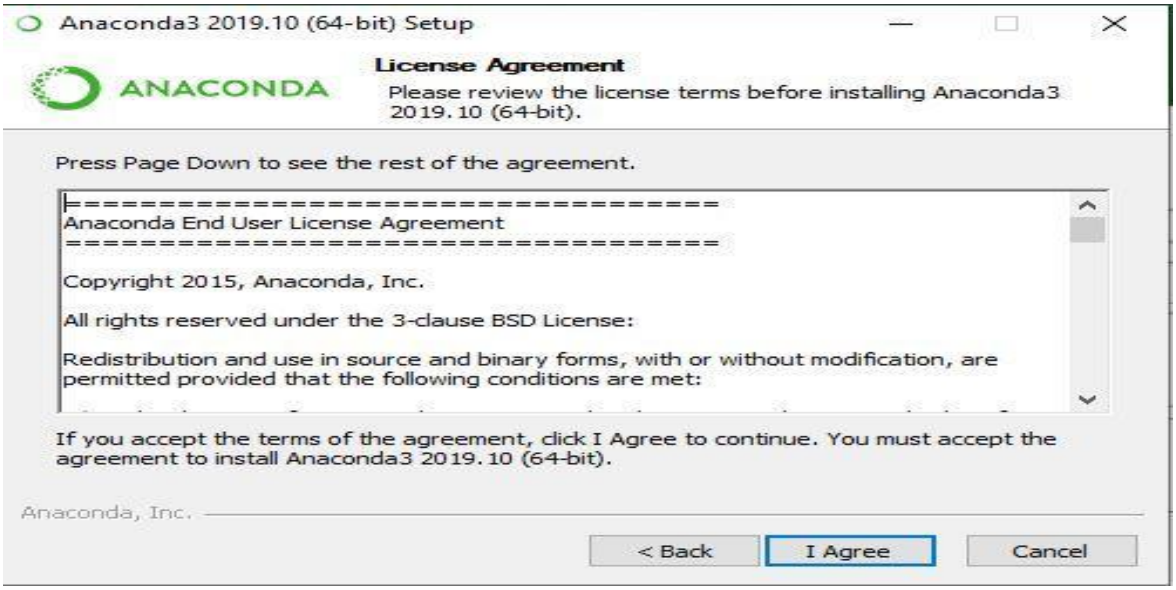
Once the installation is completed, now we will see how to setup Ananconda Installer in Windows .Begin with the installation process.



Step 4: Getting through the License Agreement :

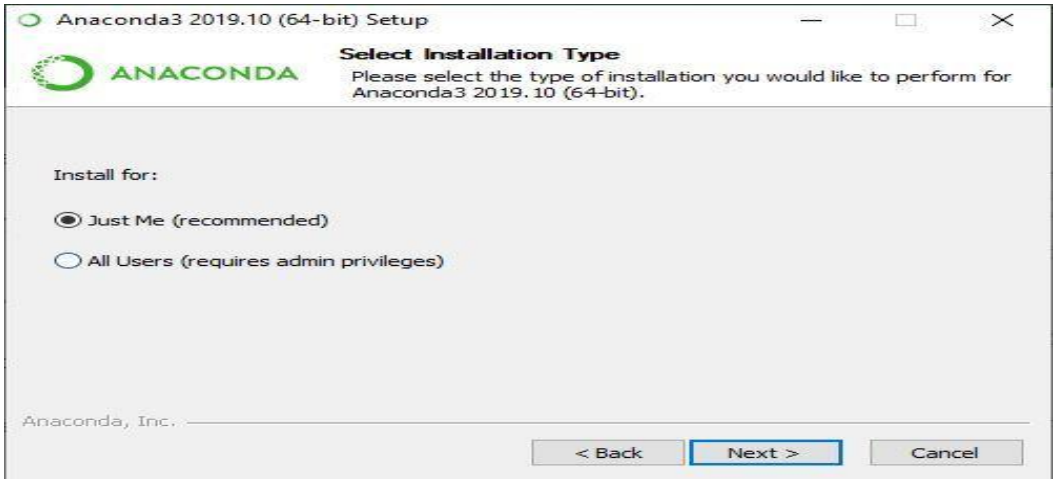
Follow the on-screen instructions, read the license terms & agreement and proceed ahead.

- Click on the **"I Agree"** button to accept the terms.
- The installation process will continue to the next setup window.



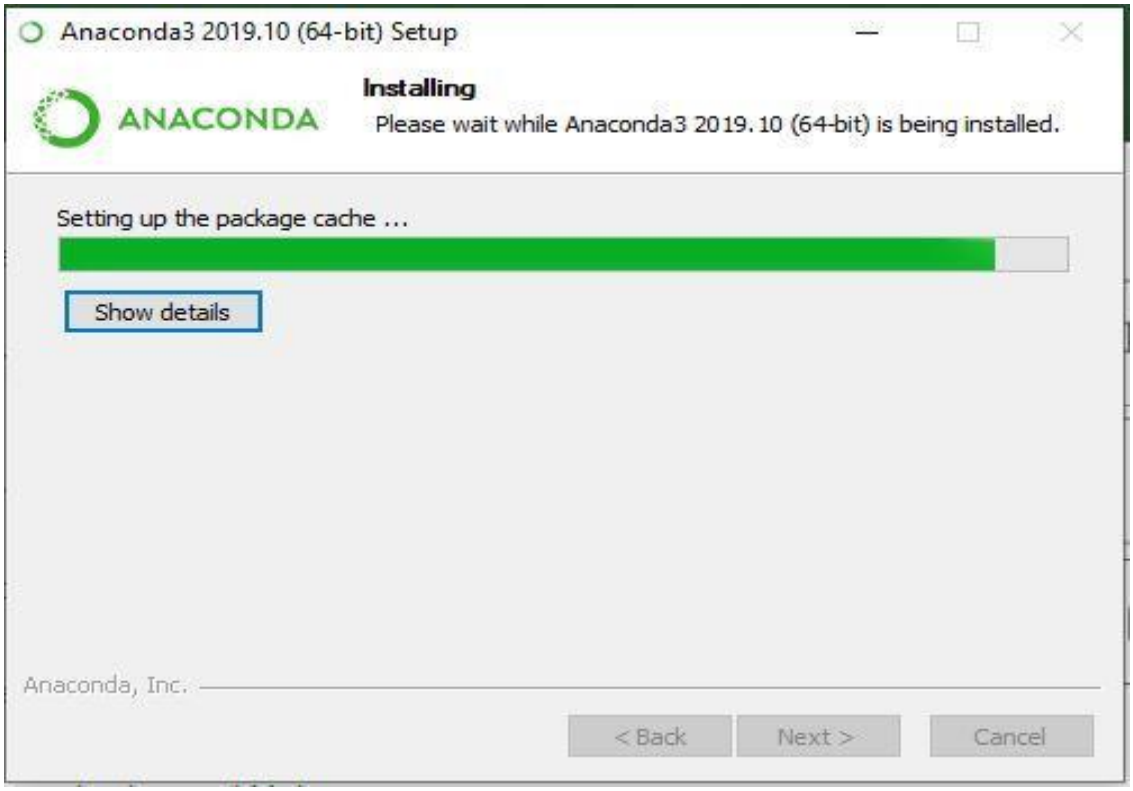
Step 5: Select Installation Type :

Select Just Me if you want the software to be used by a single User else you can select All Users.



Step 6: Getting through the Installation Process :

- Select the path where you wish to install the file extractor.
- lick "Next" to proceed .
- Click Install to start the Anaconda Installation process.



Step 7: Finishing up the Installation:

Once the installation gets complete, click Finish to complete the process.

PYCHARM

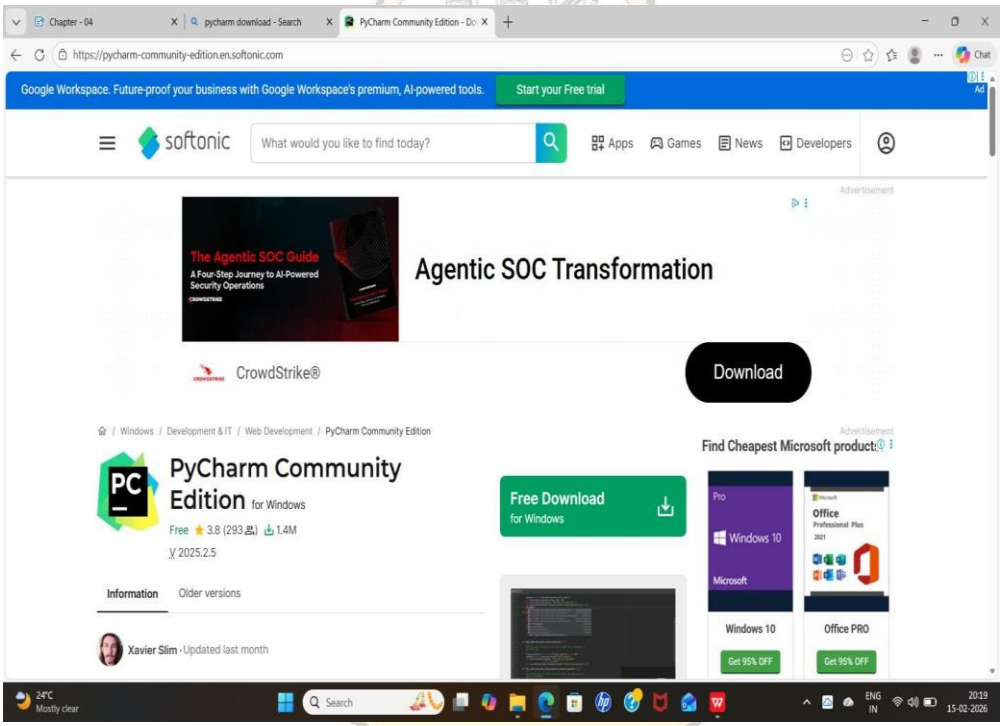
Installing PyCharm on Windows:

1. Go to the JetBrains PyCharm website

- Open any web browser (Chrome, Edge, Firefox, etc.).
- Type pycharm.jetbrains.com in the address bar and press Enter.

2. Click Download

- On the PyCharm homepage, you'll see a Download button.
- Click it to go to the download page.



3. Choose Community or Professional

You will see two versions of PyCharm:

PyCharm Community Edition (Free)

Choose this if:

- You are a beginner or student
- You are learning basic Python
- You are doing school projects, scripting, or simple programs

What it includes:

- Python editor
- Code suggestions (auto-complete)
- Debugger
- Basic project tools

4. Download the Windows (.exe) installer

- Under your chosen version, click Download for Windows.
- A file ending in .exe will start downloading.
- Wait for the download to complete.

HOW TO INSTALL THE PACKAGES :

To install the packages , open pycharm

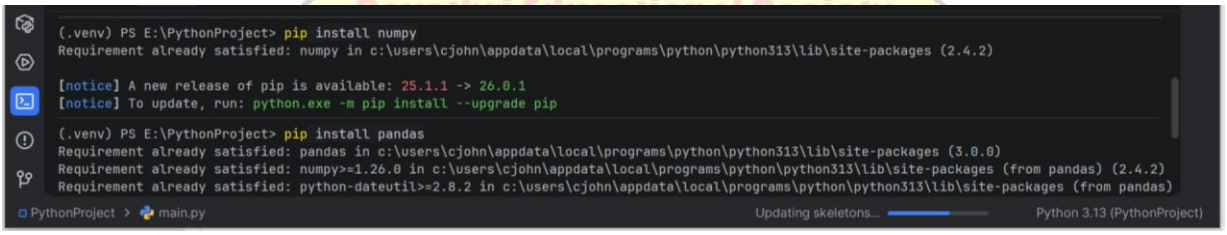
Step-1: click on Terminal .

Step-2: Now, Run the package command one by one and they will start installing in the interpreter

Python packages:

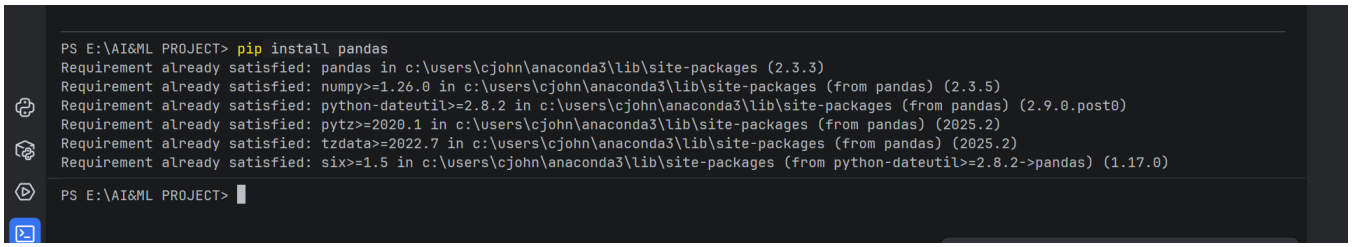
Open anaconda prompt as administrator

Type "pip install numpy" and click enter:

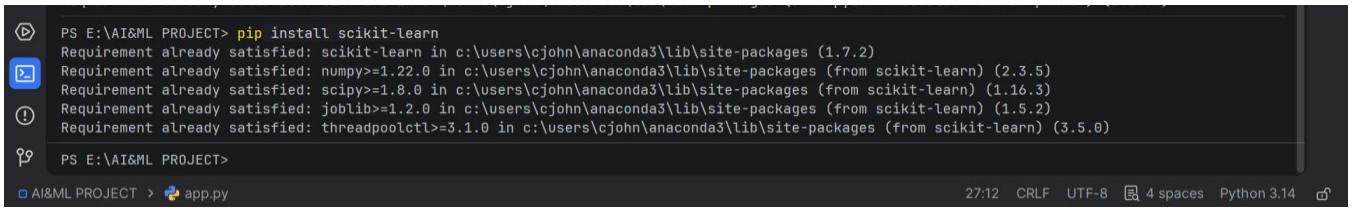


Rising Waters: A Machine Learning Approach to Flood Prediction

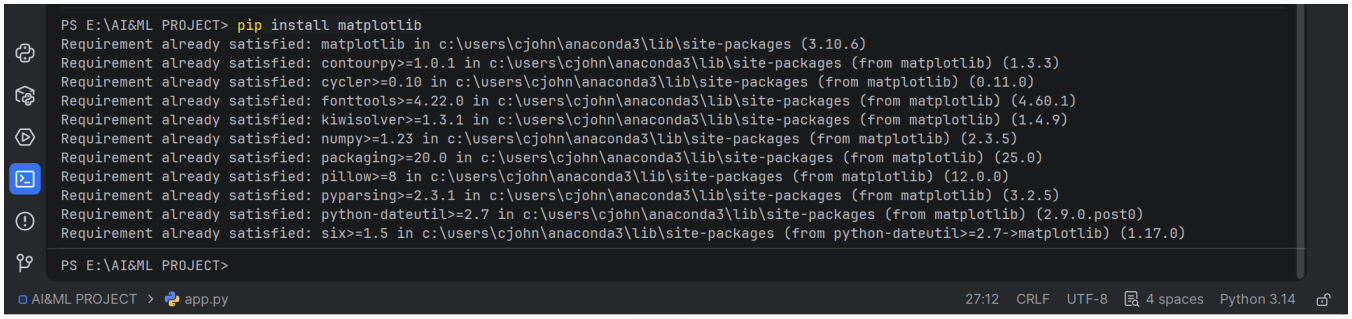
Type “pip install pandas” and click enter.



Type “pip install scikit-learn” and click enter.

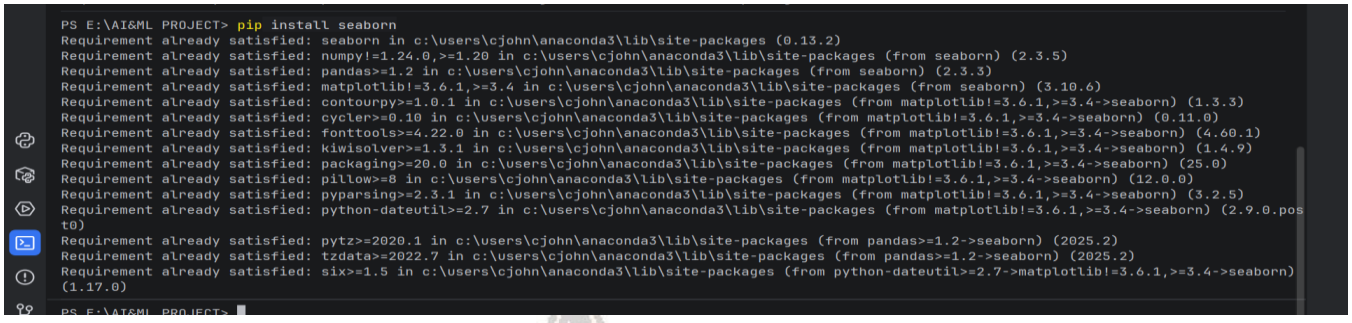


Type ”pip install matplotlib” and click enter.

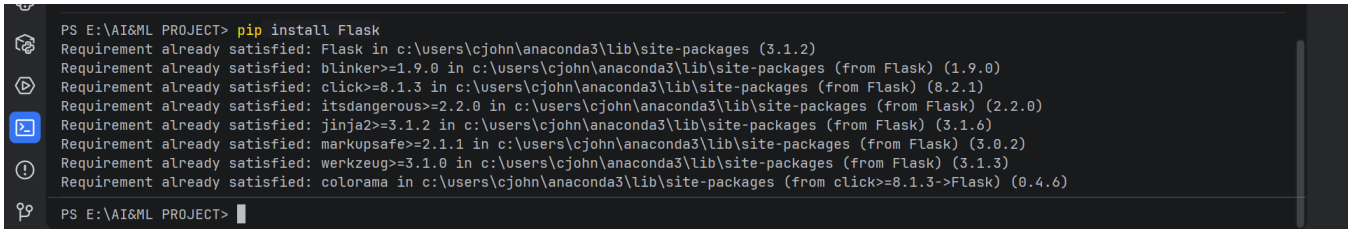


Type ”pip install pickle-mixin” and enter.

Type ”pip install seaborn” and enter.



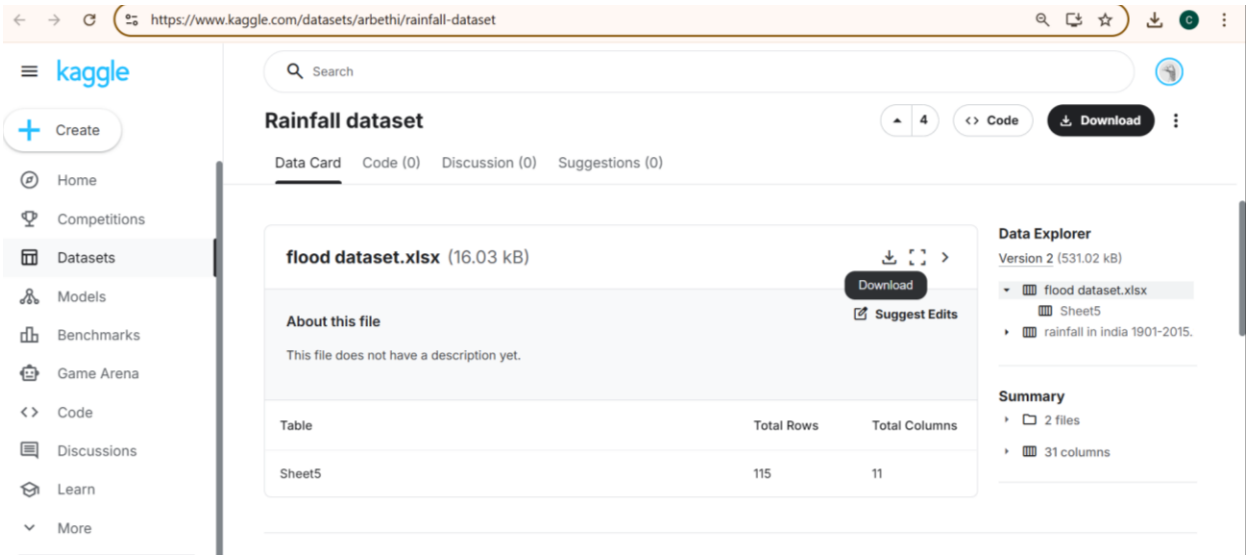
Type “pip install Flask” and enter:



DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	2 MARKS

6.2 DATA COLLECTION:

- Download the dataset from the below link
- https://www.kaggle.com/datasets/arbethi/rainfall-dataset
- Login to your Kaggle account.
- Open your dataset page (using your link or search it).
- Click the Download button (top right side).
- The file will download as a ZIP file.
- Go to your Downloads folder.
- Right-click the ZIP file → Click Extract All.
- Open the extracted folder — your dataset (CSV/Excel file) is ready

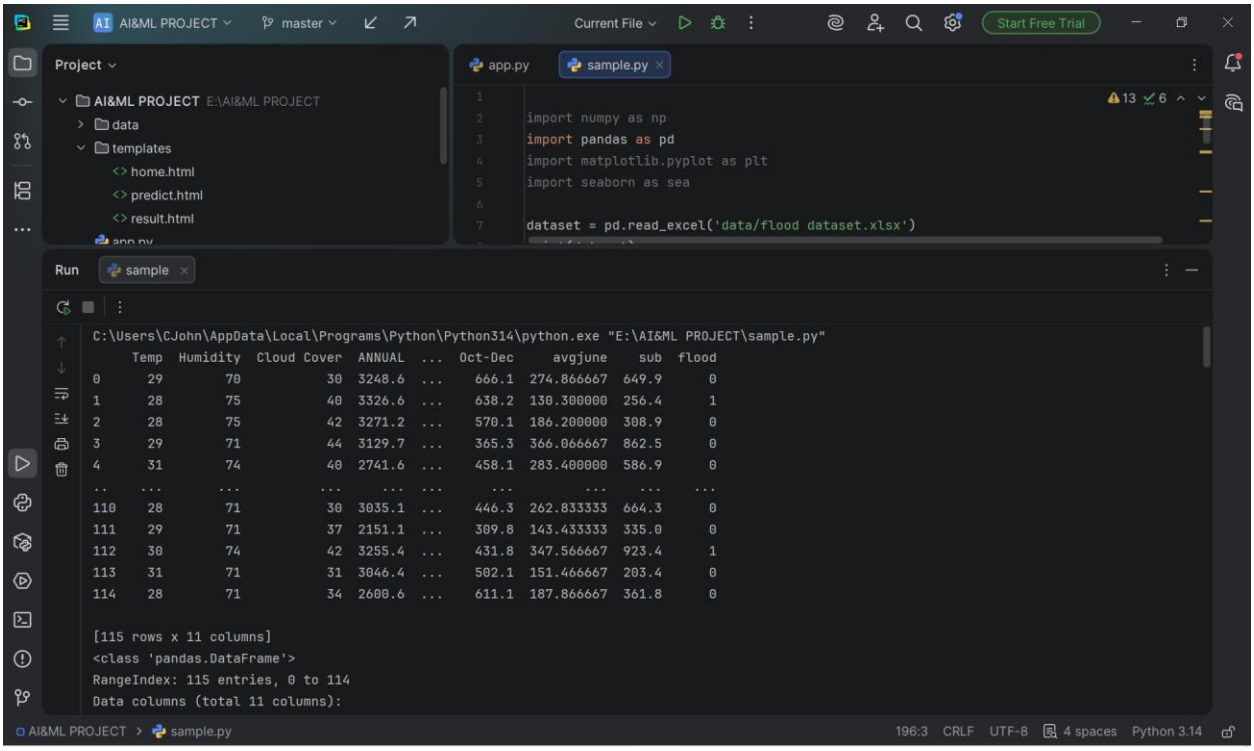


Create the folder called data and put the dataset 'flood dataset.xlsx' in it.

IMPORT THE DATASET:

Import the dataset to work by given code

```
#read the dataset
dataset=pd.read_excel('flood dataset.xlsx')
```



DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	3 MARKS

6.3 DATA PREPROCESSING:

Handling Missing Values

- Sometimes you may find some data missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.
- Word "True" that the particular column has missing values, we can also see the count of missing values in each column by using isnull().sum function.

```
#checking null values
dataset.isnull().any()
```

- isnull()

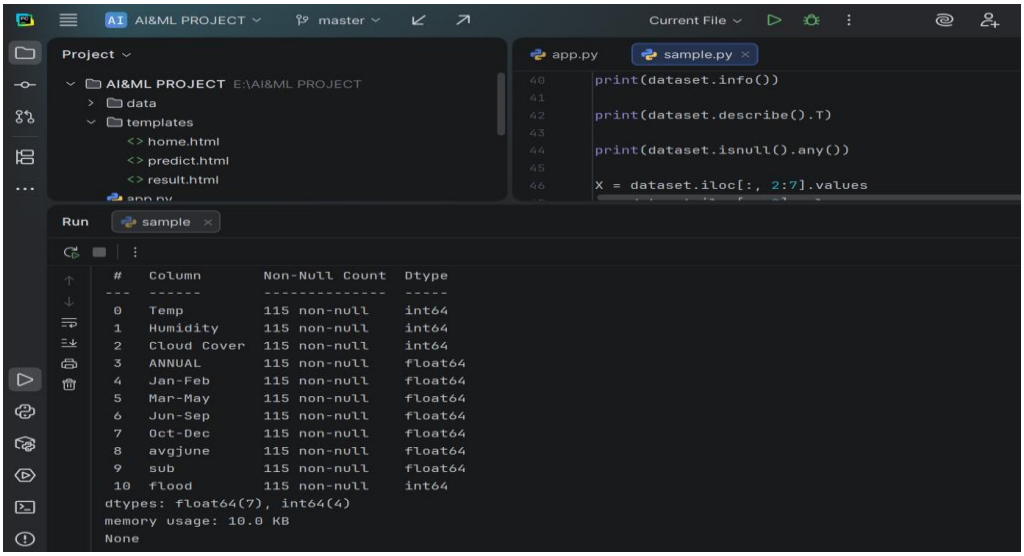
- It checks each cell in the dataset.
- If a value is missing, it returns True.
- If a value is present, it returns False.

- sum()

- It counts the number of True values in each column.
- Since True = 1 and False = 0,
- It gives the total number of missing values in each column.

- What You Understand From This

- Which columns have missing data
- How many values are missing
- Whether you should drop or fill them



Handling outliers:

With the help of boxplot, outliers are visualized (refer activity 3 univariate analysis). And here we are going to find upper bound and lower bound of Na to K feature with some mathematical formula.

- To find the upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.
- If outliers are removed, we lose more data. It will impact model performance.
- Here removing outliers is impossible. So, the capping technique is used on outliers.
- Capping: Replacing the outliers with upper bound values.

Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

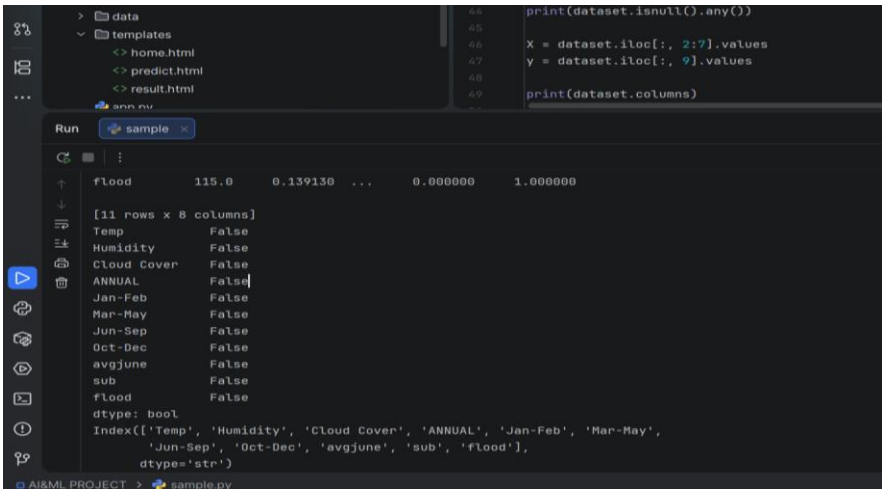
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project, we are using feature mapping and label encoding.

Splitting the Dataset into Dependent and Independent variables.

- In machine learning, the concept of the dependent variable (y) and independent variables(x) is important to understand. Here, the Dependent variable is nothing but output in the dataset and the independent variable is all inputs in the dataset. We can denote with any symbol (alphabets). In our dataset, we can say that class is the dependent variable and all other columns are independent. But in order to select the independent columns, we will be selecting only those columns which are highly correlated and some value to our dependent column.
- With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.
- Let's create out independent and dependent variables:

```
#independent features
x=dataset.iloc[:,2:7].values

#dependent feature
y=dataset.iloc[:,9:].values
```

- In the above code we are creating a Data Frame of the independent variable x with our selected columns and for the dependent variable y, we are only taking the class column.
- Where Data Frame is used to represent a table of data with rows and columns.



Split the dataset into Train set and Test set

- ·Now split our dataset into a train set and test using train_test_split class from sci-kit learn library.
- **Train_test_split:** used for splitting data arrays into training data and for testing data.

```
#split the data into train and test set from our x and y
#import train_test_split fucntion
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=10)
```

Feature scaling:

- Standard Scaler: Sklearn its main scaler, the StandardScaler, uses a strict definition of standardization to standardize data. It purely centers the data by using the following formula, where u is the mean and s is the standard deviation.

```
#import Standardscaler
from sklearn.preprocessing import StandardScaler
#create object to Standardscaler class
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

- Save the StandardScaler

```
#import dump class from joblib
from joblib import dump
dump(sc,"transform.save")
```

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	4 MARKS

6.4 MODEL BUILDING:

Decision tree model

A function named decision tree is created and train and test data are passed as the parameters. Inside the function, the DecisionTreeClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report are done.

```
In [17]: from sklearn import tree
         from sklearn import ensemble
         from sklearn import neighbors
         import xgboost

In [18]: dtree = tree.DecisionTreeClassifier()
         Rf = ensemble.RandomForestClassifier()
         knn = neighbors.KNeighborsClassifier()
         xgb = xgboost.XGBClassifier()

In [32]: dtree.fit(x_train,y_train)
         Rf.fit(x_train,y_train)
         knn.fit(x_train,y_train)
         xgb.fit(x_train,y_train)
```

How Decision Tree Works

- The model splits the dataset into branches based on feature values.
- It selects the best feature to split the data.
- The splitting continues until it reaches a final decision (leaf node).
- Final output is a predicted class (Flood or No Flood).

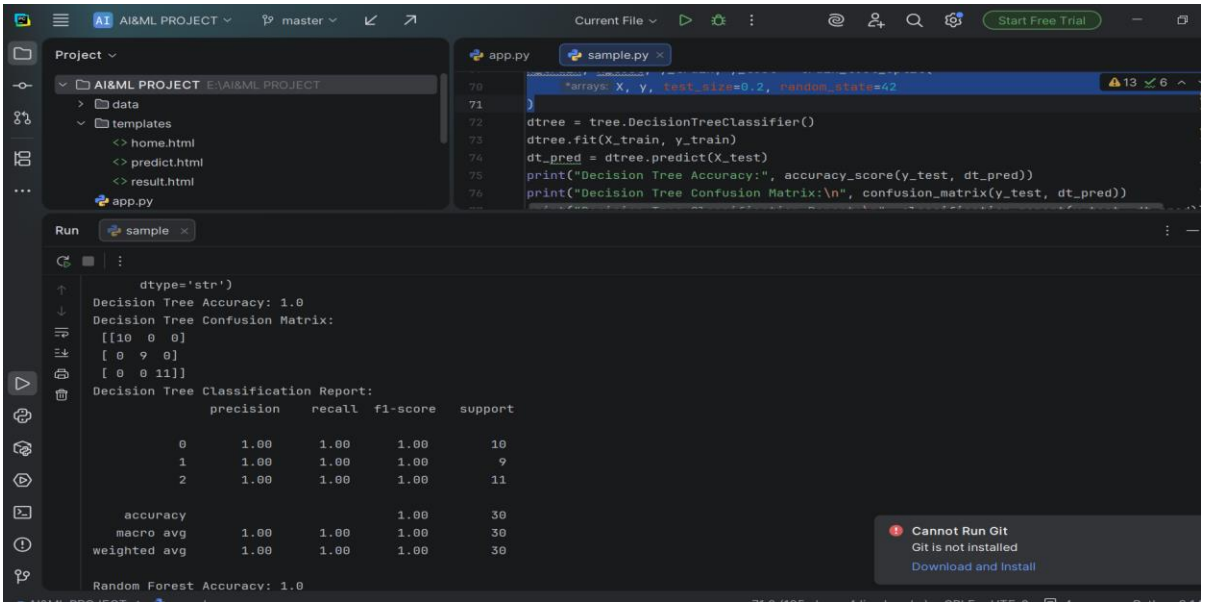
Training Process

- The training data (X_train, y_train) is given to the model.

- The model learns patterns and relationships between features and target.
- It creates rules to classify new data.

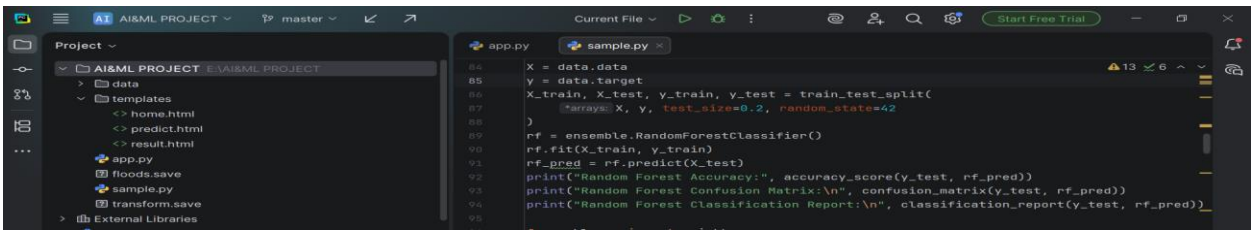
Testing Process

- Test data (X_test) is given.
- The trained model predicts output.
- Predictions are compared with actual values (y_test).



Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report are done.



What is Random Forest

Random Forest is an ensemble learning method.
It builds multiple Decision Trees and combines their results to make a final prediction.

Instead of relying on a single tree:

- Many trees are created.
- Each tree gives a prediction.
- The final output is decided by majority voting (for classification).

Function Creation

A function named randomForest is created.
The training and testing datasets are passed as parameters:

- X_train
- X_test
- y_train
- y_test

Training Process

- The RandomForestClassifier algorithm is initialized.
- The training data is passed to the model using the .fit() function.
- The model builds multiple decision trees using different subsets of data and features.
- It learns patterns from rainfall, temperature, humidity, etc.

Testing Process

- The test data is passed to the trained model.
- The .predict() function is used to generate predictions.
- The predicted values are stored in a new variable.

Model Evaluation

Confusion Matrix

Used to measure:

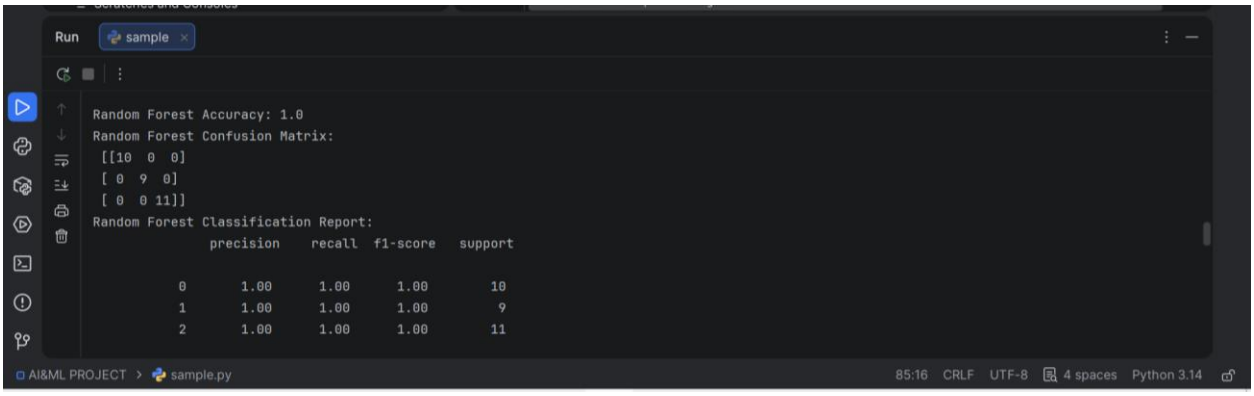
- True Positive (Correct flood prediction)
- True Negative (Correct no flood prediction)
- False Positive (Incorrect flood prediction)
- False Negative (Missed flood case)

It helps understand how well the model is performing.

Classification Report

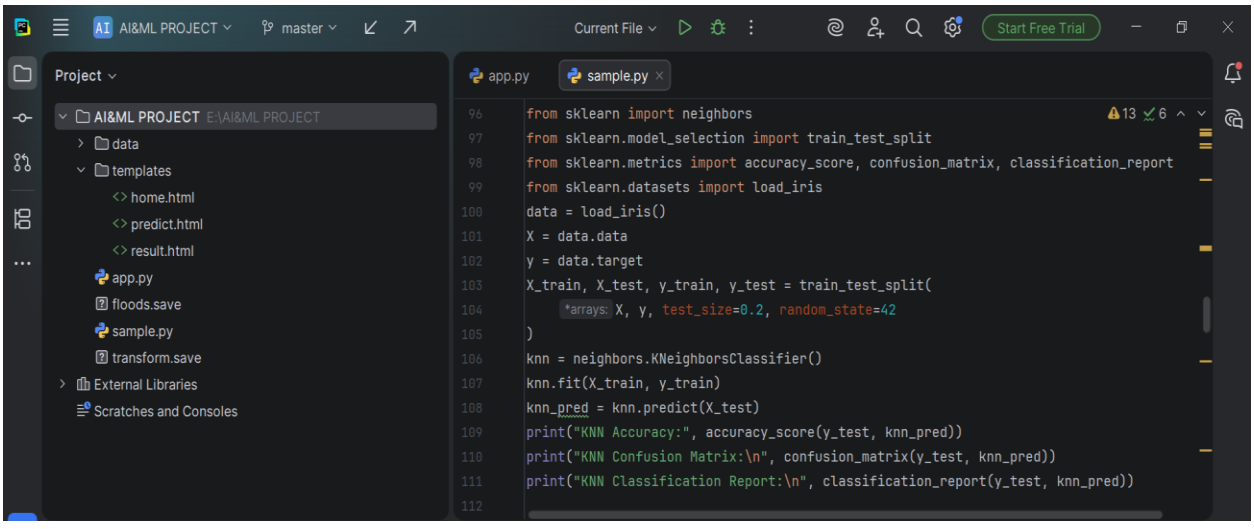
Provides performance metrics such as:

- **Accuracy** – Overall correctness of predictions
- **Precision** – Correctness of predicted flood cases
- **Recall** – Ability to detect actual flood cases
- **F1-score** – Balance between precision and recall



KNN model

A function named KNN is created and train and test data are passed as the parameters. Inside the function, the KNeighborsClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report are done.



KNN is a distance-based algorithm.
It classifies a new data point based on the K closest data points (neighbors) in the training dataset.

It works on a simple idea:

- Find the nearest K data points.
- Check their class labels.
- Assign the majority class to the new data point.

Function Creation

A function named KNN is created.
The training and testing datasets are passed as parameters:

- X_train
- X_test
- y_train
- y_test

Training Process

- The KNeighborsClassifier algorithm is initialized.
- The training data is passed using the .fit() function.
- Unlike Decision Tree or Random Forest, KNN does not build a model.
- It simply stores the training data for future comparisons.

Testing Process

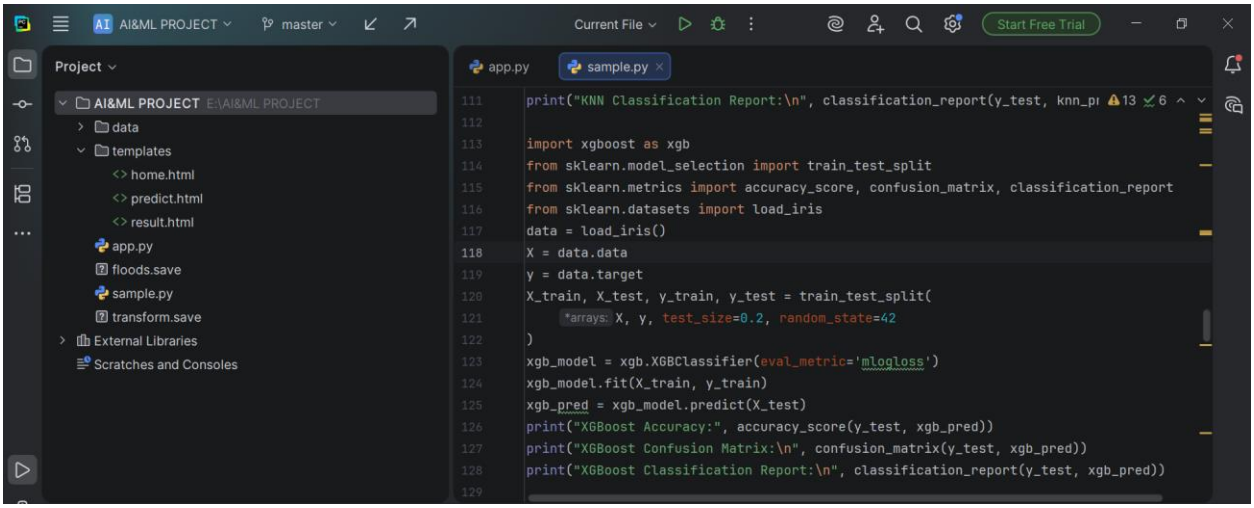
- The test data is passed to the model.
- The .predict() function calculates distances between test points and training points.
- It selects the K nearest neighbors.
- The majority class among neighbors is assigned as the prediction.
- The predicted values are stored in a new variable.

```
KNN Accuracy: 1.0
KNN Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
KNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Xgboost model

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, the GradientBoostingClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report are done.

Fit the model using x_train, y_train data



Boosting is an **ensemble learning technique** where:

- Multiple weak models (usually small decision trees) are built.
- Each new model tries to correct the errors made by the previous model.
- All models are combined to produce a strong final prediction.

Unlike Random Forest (which builds trees independently), boosting builds trees **sequentially**.

Model Initialization

Inside the function:

- The GradientBoostingClassifier algorithm is initialized.
- It creates multiple weak decision trees.
- Each tree improves the performance of the previous one.

Model Fitting (Training Phase)

The model is trained using:

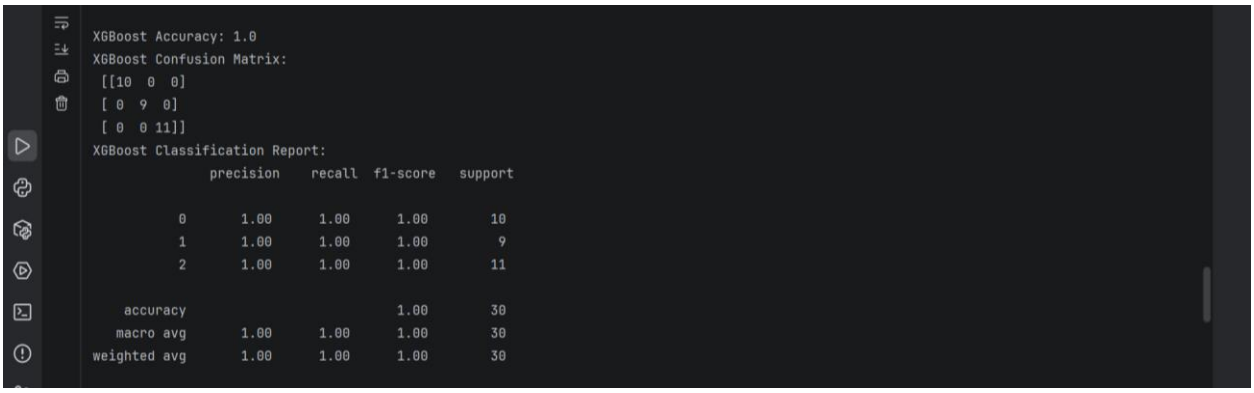
- x_train (input features)
- y_train (target variable)

Using the .fit(x_train, y_train) function:

- The model learns patterns from rainfall, humidity, temperature, etc.
- It minimizes prediction errors step by step.
- Each new tree focuses more on incorrectly predicted flood cases.

Testing Phase

- The test dataset (x_test) is given to the trained model.
- Predictions are generated using .predict(x_test).
- The predicted values are stored in a new variable.



```
XGBoost Accuracy: 1.0
XGBoost Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Compare the model

For comparing the above four models compare model function is defined.

After calling the function, the results of models are displayed as output. From the four model Decision tree, random forest and xgboost are performing well. From the below image, we can see the accuracy of the models. All three models have 96.55% accuracy

```
In [38]: from sklearn import metrics

In [39]: print(metrics.accuracy_score(y_test,p1))
         print(metrics.accuracy_score(y_test,p2))
         print(metrics.accuracy_score(y_test,p3))
         print(metrics.accuracy_score(y_test,p4))

          0.9655172413793104
          0.9655172413793104
          0.896551724137931
          0.9655172413793104
```

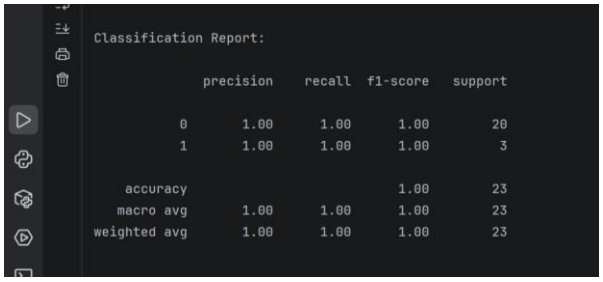
OUTPUT:

Evaluating performance of the model

- After comparing all the three models with different attributes ,xgboost is the better model,so we will save this model

```
In [41]: metrics.confusion_matrix(y_test,p4)

         array([[25,  1],
                [ 0,  3]], dtype=int64)

In [40]: print(metrics.accuracy_score(y_test,p4))

         0.9655172413793104

In [42]: print(metrics.precision_score(y_test,p4))

         0.75

In [43]: print(metrics.recall_score(y_test,p4))

         1.0
```

OUTPUT:



```
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00         3

    accuracy                           1.00        23
   macro avg       1.00      1.00      1.00        23
weighted avg       1.00      1.00      1.00        23
```

Saving the model

- Joblib of save is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.
- Save our model by importing joblib dump class.

```
#saving the file
from joblib import dump
dump(xg_cla,'floods.save')

 ['floods.save']
```

Here, xg_clas is our Xgboost Classifier with saving as floods. save file.

FULL CODE OF MODEL BUILDING:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sea

dataset = pd.read_excel('data/flood dataset.xlsx')
print(dataset)

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(dataset['Temp'], kde=True)
plt.show()

sns.boxplot(x=dataset['Temp'])
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.gcf()
fig.set_size_inches(15, 15)
```

```
fig = sns.heatmap(
    dataset.corr(),
    annot=True,
    cmap='summer',
    linewidths=1,
    linecolor='k',
    square=True,
    mask=False,
    vmin=-1,
    vmax=1,
    cbar_kws={"orientation": "vertical"},
    cbar=True
)
plt.show()


dataset.head()


print(dataset.info())


print(dataset.describe().T)


print(dataset.isnull().any())
```

```
X = dataset.iloc[:, 2:7].values

y = dataset.iloc[:, 9].values



print(dataset.columns)



from sklearn.model_selection import train_test_split

X_train, X_test = train_test_split(dataset, test_size=0.25,
random_state=10)



from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)



from joblib import dump

dump(sc, "transform.save")



from sklearn import tree

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data
```

```
y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

dtree = tree.DecisionTreeClassifier()

dtree.fit(X_train, y_train)

dt_pred = dtree.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, dt_pred))

print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test,
dt_pred))

print("Decision Tree Classification Report:\n",
classification_report(y_test, dt_pred))



from sklearn import ensemble

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)
```

```
rf = ensemble.RandomForestClassifier()

rf.fit(X_train, y_train)

rf_pred = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))

print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test,
rf_pred))

print("Random Forest Classification Report:\n",
classification_report(y_test, rf_pred))



from sklearn import neighbors

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

knn = neighbors.KNeighborsClassifier()

knn.fit(X_train, y_train)

knn_pred = knn.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, knn_pred))
```

```
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))

print("KNN Classification Report:\n", classification_report(y_test,
knn_pred))



import xgboost as xgb

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

xgb_model = xgb.XGBClassifier(eval_metric='mlogloss')

xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)

print("XGBoost Accuracy:", accuracy_score(y_test, xgb_pred))

print("XGBoost Confusion Matrix:\n", confusion_matrix(y_test,
xgb_pred))

print("XGBoost Classification Report:\n",
classification_report(y_test, xgb_pred))



# Import libraries
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model1 = LogisticRegression(max_iter=200)
model1.fit(X_train, y_train)
p1 = model1.predict(X_test)
model2 = DecisionTreeClassifier()
model2.fit(X_train, y_train)
p2 = model2.predict(X_test)
model3 = RandomForestClassifier()
model3.fit(X_train, y_train)
p3 = model3.predict(X_test)
```

```
model4 = SVC()
model4.fit(X_train, y_train)
p4 = model4.predict(X_test)
print("Model 1 Accuracy:", metrics.accuracy_score(y_test, p1))
print("Model 2 Accuracy:", metrics.accuracy_score(y_test, p2))
print("Model 3 Accuracy:", metrics.accuracy_score(y_test, p3))
print("Model 4 Accuracy:", metrics.accuracy_score(y_test, p4))


from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
data = load_breast_cancer()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=29, random_state=2
)
model = LogisticRegression(max_iter=5000)
model.fit(X_train, y_train)
p4 = model.predict(X_test)
print(metrics.confusion_matrix(y_test, p4))
print(metrics.accuracy_score(y_test, p4))
```

```
print(metrics.precision_score(y_test, p4))

print(metrics.recall_score(y_test, p4))


import pandas as pd

import joblib

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

df = pd.read_excel('data/flood dataset.xlsx')

if df["flood"].dtype == object:

    df["flood"] = df["flood"].map({"Yes": 1, "No": 0})

print("Flood value counts:")

print(df["flood"].value_counts())

X = df[[

    "Cloud Cover",

    "ANNUAL",

    "Jan-Feb",

    "Mar-May",

    "Jun-Sep"

]]

y = df["flood"]

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42
```

```
)
model = RandomForestClassifier(
    n_estimators=500,
    max_depth=15,
    class_weight='balanced',
    random_state=42
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
joblib.dump(model, "floods.save")
print("\nModel retrained successfully!")
```

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	4 MARKS

6.5 APPLICATION BUILDING:

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

Building HTML Pages:

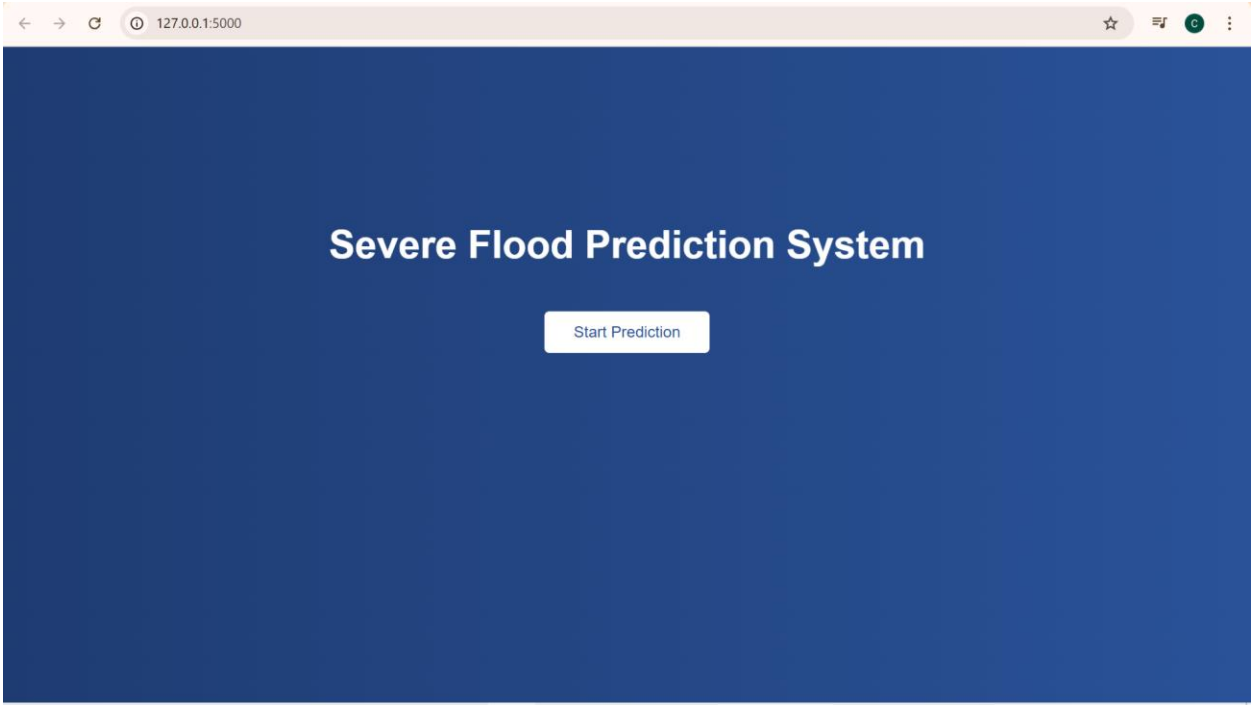
- Flask Frame Work with Machine Learning Model In this section we will be building a web application which is integrated to the model we built. An UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
- Previously we are saved this file as “floods.save”. We have 5 independent variables and one dependent variable for this model.
- To build this you should know the basics of “HTML, CSS, Bootstrap, flask framework and python” Create a project folder that should contain.
- A python file called app.py.
- Model file (floods.save).
- Templates folder which contains index.HTML file.
- Static folder which contains CSS folder which contains styles.css.

We use HTML to create the front-end part of the web page.

Here, we created 4 html pages- home.html, image.html, imageprediction.html, intro.html.

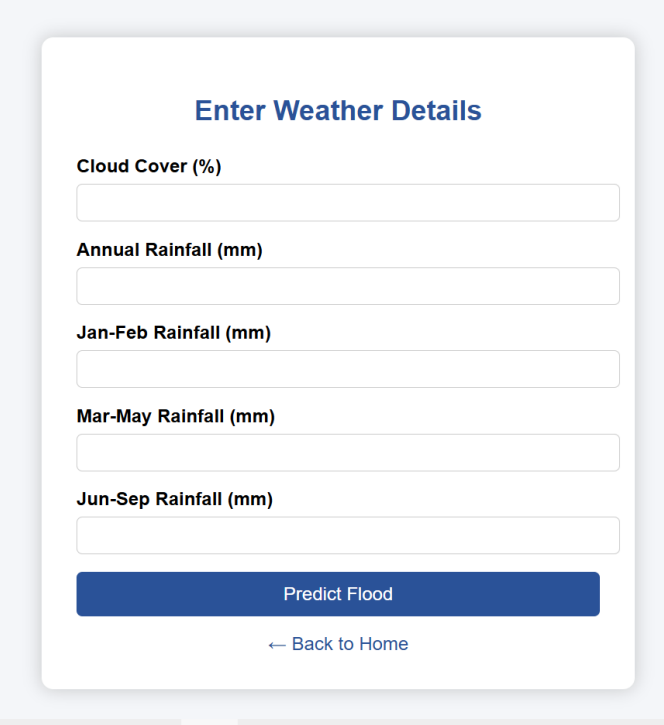
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

- This is your **main page**
- Shows project title
- Has "Start Prediction" button
- Redirects to prediction form page



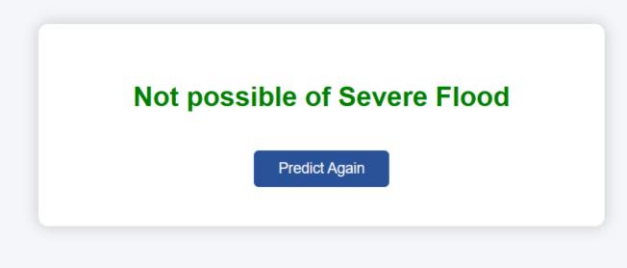
- This page collects user input
- Takes 5 independent variables:

    - Cloud Cover
    - Annual Rainfall
    - Jan-Feb Rainfall
    - Mar-May Rainfall
    - Jun-Sep Rainfall



Displays prediction result
✓ Shows:

- Prediction message
- Colour (Red for Flood, Green for No Flood)
- Has "Predict Again" button



Build python code:

- Let us build flask file 'app.py' which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.
- App starts running when "__name__" constructor is called in main.

- render_template is used to return HTML file.
- "GET" method is used to take input from the user.
- "POST" method is used to display the output to the user.
- We will be using python for server side scripting. Let's see step by step process for writing backend code.

Importing Libraries

- Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

```
from flask import Flask, render_template, request
# used to run/serve our application
# render_template is used for rendering the html pages
#import load from joblib to load the saved model file
from joblib import load
```

Create Flask app and Load our model file

```
app=Flask(__name__) # our flask app
#load model file
model =load('floods.save')
sc=load('transform.save')
```

Routing to the HTML Page

Here we will be using the declared constructor to route to the HTML page that we have created earlier.

In the above example, the '/' URL is bound with the home.html function. Hence, when the home page of the webserver is opened in the browser, the HTML page is rendered. Whenever you click on the Intro button from the home page, the intro.html page will be rendered.

```
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")
```

We are routing the app to the HTML templates which we want to render. Firstly we are rendering the "home.html" template which is the home page to our web UI. Where it will display two options, one is Home and Predict Floods.
When you click on Predict Floods, it redirects to the next page which is bounded with URL /predict. At that time index.html page will be rendered. Where to have to fill in all the details and get the result on the prediction page.

Route the prediction on UI

- predict() – is taking the values from the prediction page and storing it into a variable and then we are creating a DataFrame along with the values and 5 independent features and finally, we are predicting the values using or loaded model which we build and storing the output in a variable and returning it to the result page.

```
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    temp = request.form['temp']
    Hum = request.form['Hum']
    db = request.form['db']
    ap = request.form['ap']
    aa1 = request.form['aa1']

    data = [[float(temp),float(Hum),float(db),float(ap),float(aa1)]]
    prediction = model.predict(sc.transform(data))
    output=prediction[0]
    if(output==0):
        return render_template('noChance.html', prediction='No possibility of severe flood')
    else:
        return render_template('chance.html', prediction='possibility of severe flood')
```

Main Function

- This is used to run the application in localhost

```
if __name__ == '__main__':
    app.run(debug=True)
```

FULL PYTHON CODE:

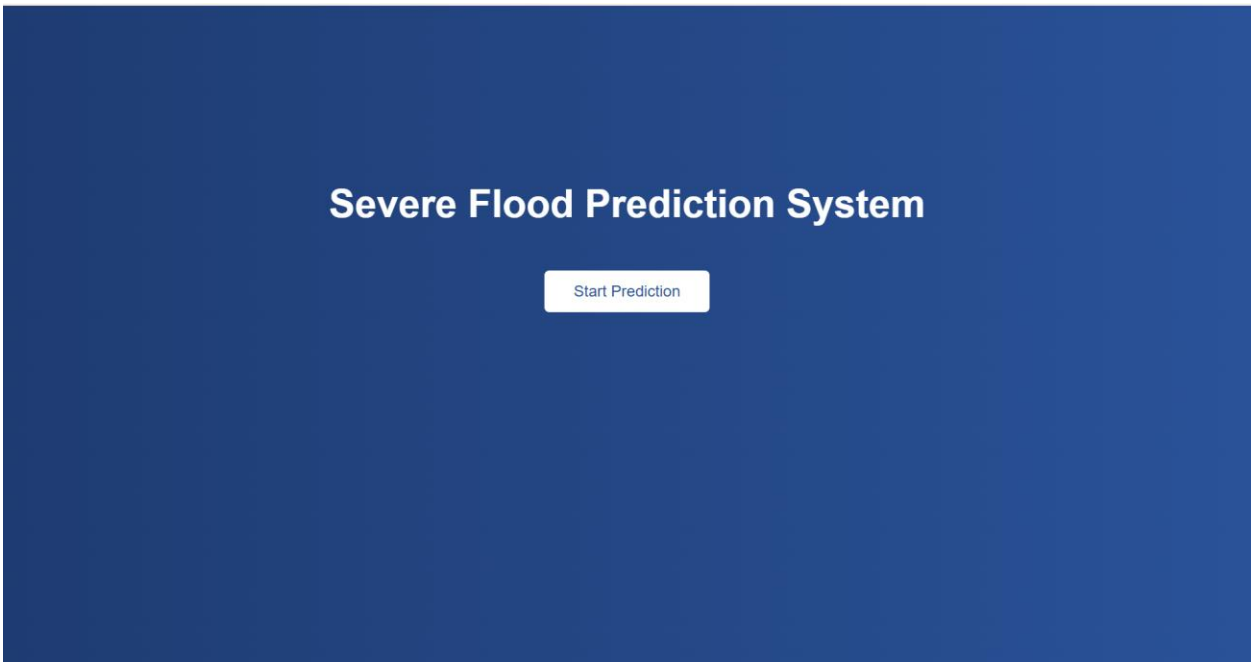
```
from flask import Flask, render_template, request
import pandas as pd
import joblib
app = Flask(__name__)
model = joblib.load("floods.save")
@app.route('/')
def home():
    return render_template("home.html")
@app.route('/predict_page')
def predict_page():
    return render_template("predict.html")
@app.route('/predict', methods=['POST'])
def predict():
    try:
        cloud = float(request.form["cloud_cover"])
        annual = float(request.form["annual"])
        jan_feb = float(request.form["jan_feb"])
        mar_may = float(request.form["mar_may"])
```

```
        jun_sep = float(request.form["jun_sep"])


        input_data = pd.DataFrame([{
            "Cloud Cover": cloud,
            "ANNUAL": annual,
            "Jan-Feb": jan_feb,
            "Mar-May": mar_may,
            "Jun-Sep": jun_sep
        }])
        prediction = model.predict(input_data)
        if prediction[0] == 1:
            result = "Possible of Severe Flood"
            color = "red"
        else:
            result = "Not possible of Severe Flood"
            color = "green"
        return render_template("result.html",
                               prediction=result,
                               color=color)
    except Exception as e:
        return f"Error: {str(e)}"


if __name__ == "__main__":
```

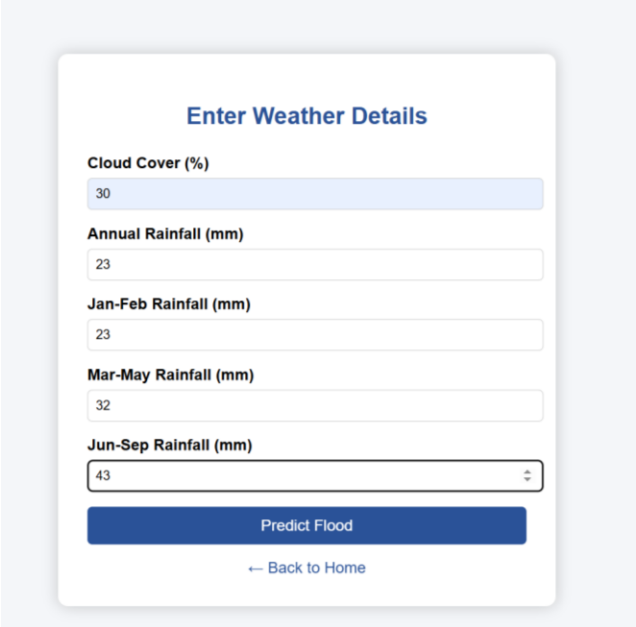
```
app.run(debug=True)
```

Output:

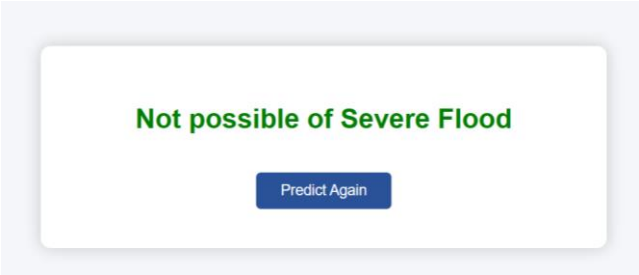
- This is the home page of floods prediction.



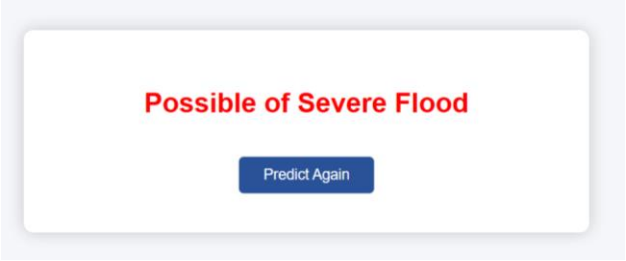
Then Click on the predict floods which redirect to the prediction page, user gives the input for predicting the output where they can give input as Cloud visibility, Annual Rainfall, some other inputs, then click to submit the output



OUTPUT-1:



OUTPUT-2:



DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	5 MARKS

7.1 FUNCTIONAL AND PERFORMANCE TESTING:

◆ **Functional Testing**

- Input validation (no empty/invalid values)
- Correct flood risk prediction (Low / Moderate / High)
- Accuracy evaluation (Accuracy, Precision, Recall, F1-score)
- Proper result display in Flask web app
- Alert colour coding works correctly

◆ **Performance Testing**

- Response time within 1–2 seconds
- Handles multiple user requests
- Stable under extreme input values
- Scalable for district/state-level deployment

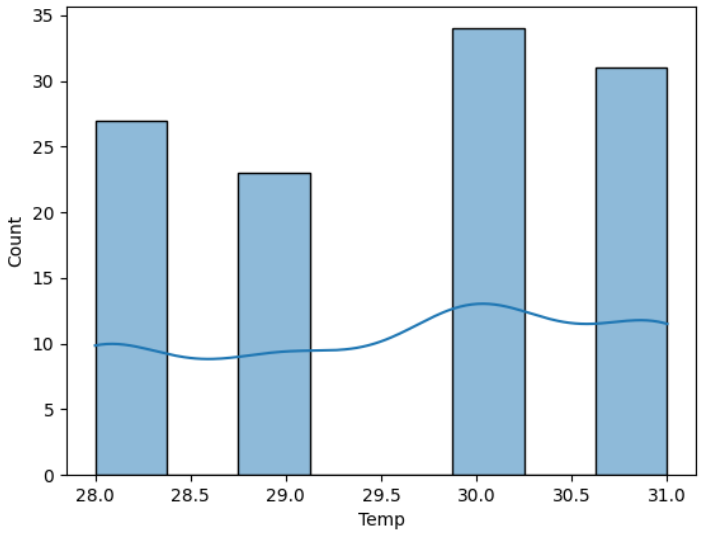
✓ **Result**

- System predicts flood risk accurately
- Web application runs smoothly
- No crashes under normal load
- Suitable for real-time flood warning use

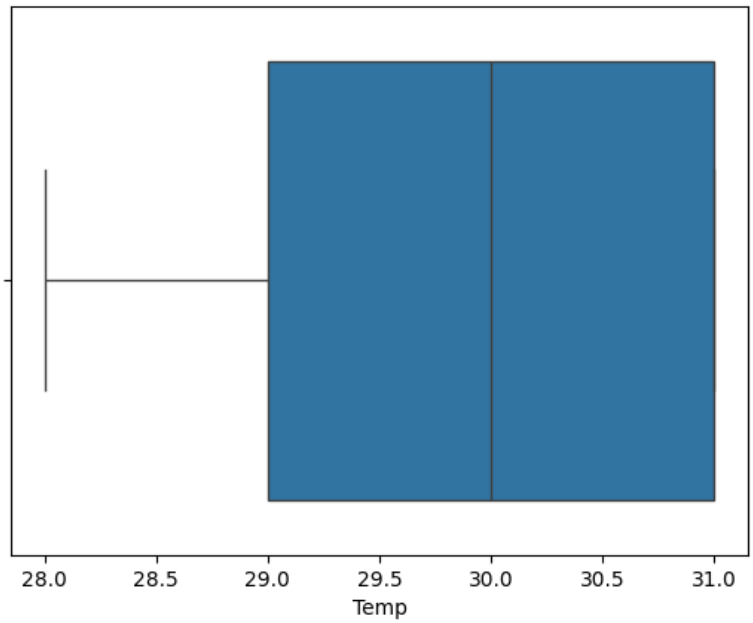
CHAPTER-8

8.1 OUTPUT SCREENS:

- **Histogram Showing Frequency Distribution of Temperature**

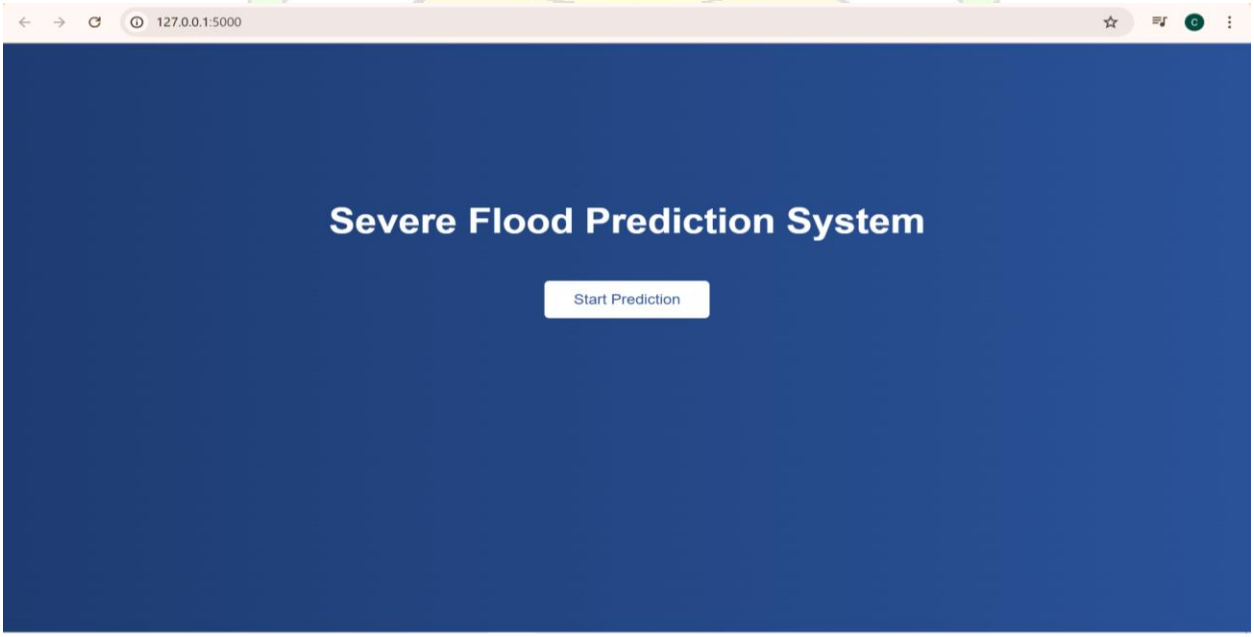


- **Box Plot Showing Distribution of Temperature Value**

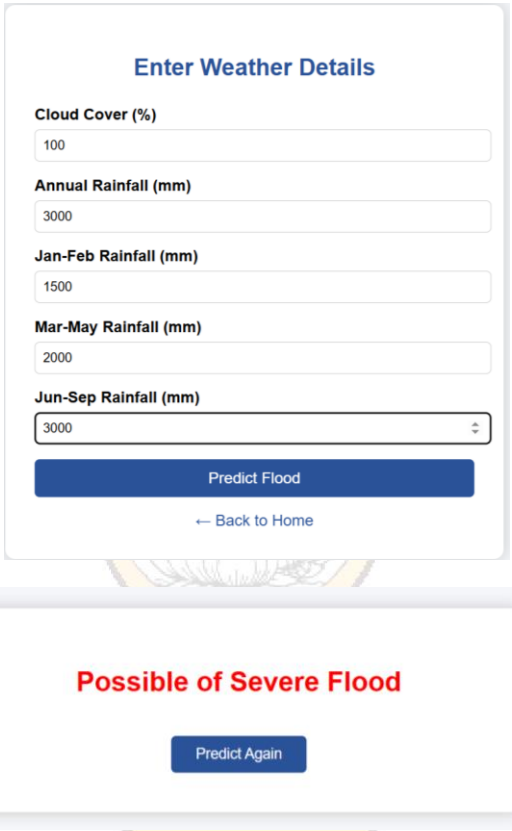


➢ **Correlation Matrix of Climatic and Rainfall Features for Flood Prediction**

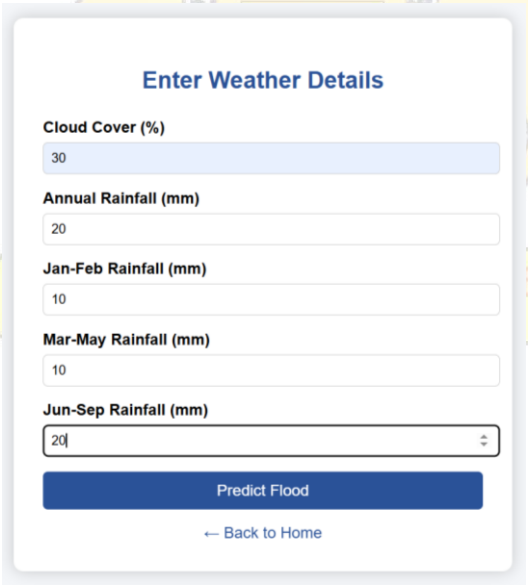
- **Flood Prediction Result**

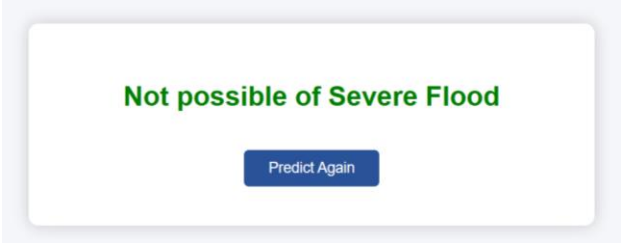


- **High Flood Risk Detected**



- **No Flood Risk Detected**





CHAPTER-9

9.1 ADVANTAGES:

Early Warning System

The system provides advance flood alerts based on environmental data, helping authorities and communities take preventive measures before disaster strikes. This reduces loss of life and property damage.

High Prediction Accuracy

By using advanced machine learning algorithms such as Random Forest and XGBoost, the system improves forecasting accuracy compared to traditional manual methods.

Real-Time Risk Assessment

The web-based application can instantly analyse rainfall, river level, and other parameters to provide immediate flood risk predictions.

Data-Driven Decision Making

Government agencies and disaster management authorities can rely on accurate data analysis to make informed decisions about evacuation and emergency response.

Cost-Effective Solution

Automated flood prediction reduces the need for constant manual monitoring and expensive infrastructure, making it an economical solution.

Scalable Architecture

The system can be expanded to cover multiple districts, states, or even nationwide implementation. It can also integrate with IoT sensors for real-time data collection.

User-Friendly Interface

The Flask-based web application provides a simple interface where users can input environmental parameters and receive flood risk levels easily.

Reduced Human Error

Automation minimizes manual calculation errors and improves prediction reliability.

9.2 DISADVANTAGE:

Dependence on Data Quality

The system's accuracy depends on the quality and availability of historical and real-time data. Incomplete or incorrect data can reduce prediction accuracy.

Limited Prediction Accuracy

Machine learning models cannot guarantee 100% accurate predictions. Unexpected natural events may affect results.

Requires Continuous Data Updates

To maintain accuracy, the model must be retrained regularly with updated environmental data.

Infrastructure Dependency

The system requires internet connectivity, server availability, and proper deployment infrastructure for smooth operation.

High Initial Development Effort

Developing, training, and testing machine learning models requires technical expertise and time.

Cannot Replace Human Expertise Completely

The system supports decision-making but cannot fully replace meteorologists and disaster management experts.

Risk of False Alerts

Incorrect predictions may cause:

- False positives (unnecessary panic)
- False negatives (missed warnings)

CHAPTER-10

10.1 CONCLUSION:

The Rising Waters project presents an intelligent and data-driven approach to flood prediction using machine learning techniques. By analysing environmental factors such as rainfall, river levels, humidity, and temperature, the system can predict flood risk levels with improved accuracy and speed.

The proposed solution integrates data pre-processing, feature engineering, model training, and deployment through a Flask-based web application. This ensures real-time prediction capability and user-friendly access for both authorities and individuals.

The system enhances disaster preparedness by providing early warnings, supporting informed decision-making, and reducing potential loss of life and property. Although the model depends on data quality and continuous updates, it offers a scalable and efficient solution for modern flood risk management.

In conclusion, the project demonstrates how machine learning can be effectively applied to environmental disaster prediction, contributing to smarter and more proactive flood management systems.

CHAPTER-11

11.1 FUTURE SCOPE:

Integration with IoT Sensors

The system can be connected to real-time IoT sensors to automatically collect rainfall, river level, and soil moisture data for more accurate live predictions.

Real-Time Satellite Data Integration

Future versions can integrate satellite-based weather monitoring systems for better rainfall and storm tracking.

Mobile Application Development

A dedicated mobile app can be developed to provide instant flood alerts and notifications to users in affected regions.

Deep Learning Implementation

Advanced models such as LSTM (Long Short-Term Memory) networks can be used for time-series forecasting to improve prediction accuracy.

GIS-Based Flood Mapping

Integration with Geographic Information Systems (GIS) can help visualize flood-prone areas on interactive maps.

Government & Disaster Management Integration

The system can be connected with official agencies like the National Disaster Management Authority for coordinated emergency response.

Multi-Region Deployment

The architecture can be scaled to support multiple districts, states, or nationwide flood monitoring systems.

Automated Alert Systems

Future enhancements may include:

- SMS alerts
- Email notifications

CHAPTER-12

12.1 SOURCE CODE:

Home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Severe Flood Prediction</title>
    <style>
        body {
            margin: 0;
            font-family: Arial, sans-serif;
            background: linear-gradient(to right, #1e3c72, #2a5298);
            color: white;
            text-align: center;
        }

        .container {
            margin-top: 180px;
        }

        h1 {
            font-size: 40px;
        }

        button {
            padding: 12px 30px;
            font-size: 16px;
            border: none;
            border-radius: 5px;
            background-color: #ffffff;
            color: #2a5298;
            cursor: pointer;
        }

        button:hover {
            background-color: #ddd;
        }
    </style>
</head>
<body>
```

```
<div class="container">
    <h1>Severe Flood Prediction System</h1>
    <br>
    <a href="/predict_page">
        <button>Start Prediction</button>
    </a>
</div>

</body>
</html>
```

Predict.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Enter Weather Details</title>

    <style>
        body {
            margin: 0;
            font-family: Arial, sans-serif;
            background: #f4f6f9;
        }

        .form-container {
            width: 450px;
            margin: 80px auto;
            background: white;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0,0,0,0.2);
        }

        h2 {
            text-align: center;
            margin-bottom: 25px;
            color: #2a5298;
        }

        label {
```

```
            font-weight: bold;
        }

        input {
            width: 100%;
            padding: 8px;
            margin: 6px 0 15px 0;
            border-radius: 5px;
            border: 1px solid #ccc;
        }

        button {
            width: 100%;
            padding: 10px;
            border: none;
            background-color: #2a5298;
            color: white;
            font-size: 16px;
            border-radius: 5px;
            cursor: pointer;
        }

        button:hover {
            background-color: #1e3c72;
        }

        .back {
            margin-top: 15px;
            text-align: center;
        }

        .back a {
            text-decoration: none;
            color: #2a5298;
        }

    </style>
</head>

<body>

<div class="form-container">
```

```
    <h2>Enter Weather Details</h2>

    <form action="/predict" method="POST">

        <label>Cloud Cover (%)</label>
        <input type="number" name="cloud_cover" step="any" required>

        <label>Annual Rainfall (mm)</label>
        <input type="number" name="annual" step="any" required>

        <label>Jan-Feb Rainfall (mm)</label>
        <input type="number" name="jan_feb" step="any" required>

        <label>Mar-May Rainfall (mm)</label>
        <input type="number" name="mar_may" step="any" required>

        <label>Jun-Sep Rainfall (mm)</label>
        <input type="number" name="jun_sep" step="any" required>

        <button type="submit">Predict Flood</button>

    </form>

    <div class="back">
        <a href="/">← Back to Home</a>
    </div>

</div>

</body>
</html>
```

Result.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>

    <style>
        body {
            margin: 0;
            font-family: Arial, sans-serif;
```

```
            background: #f4f6f9;
            text-align: center;
        }

        .result-box {
            width: 500px;
            margin: 150px auto;
            padding: 40px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0,0,0,0.2);
            background-color: white;
        }

        h1 {
            font-size: 28px;
        }

        .btn {
            margin-top: 20px;
            padding: 10px 25px;
            border: none;
            background-color: #2a5298;
            color: white;
            font-size: 15px;
            border-radius: 5px;
            cursor: pointer;
        }

        .btn:hover {
            background-color: #1e3c72;
        }

    </style>
</head>

<body>

<div class="result-box">

    <h1 style="color: {{color}};">
        {{prediction}}
    </h1>
```

```
    <a href="/predict_page">
        <button class="btn">Predict Again</button>
    </a>

</div>

</body>
</html>
```

Dataset link: https://www.kaggle.com/datasets/arbethi/rainfall-dataset

Dataset QR code:

GitHub link:

https://github.com/chitti4569/LTVIP2026TMIDS89043-Victorjoy/tree/master

GitHub QR code:

Demo link:
https://drive.google.com/file/d/1do3d5Iyx1TwVpo1La1qL3cWBNQGtWdkC/view?usp=drive_link

Demo QR code: