# FLOOD MONITORING SYSTEM

# TEAM MEMBER

# 911721104014 : S.BALAGANESH

## PHASE-5 DOCUMENT SUBMISSION

## Introduction:

The Flood Monitoring System is designed to provide timely flood warnings, enhance preparedness, and protect lives and property. This project spans multiple phases, starting from project definition and culminating in system deployment and integration. Each phase contributes to the development of a comprehensive flood monitoring system.

**Project Title: IoT-Based Flood Monitoring and Early Warning System**

## Objective:

The project aims to enhance flood preparedness and response by deploying IoT sensors near water bodies and flood-prone areas to monitor water levels. It intends to provide timely flood warnings to the public and emergency response teams through a user-friendly public platform.

## Phase 1: Problem Definition and Design Thinking

### Design Thinking:

Scope of the IoT-Based Flood Monitoring and Early Warning System:

Define the scope of the project's capabilities, including:

Real-time Water Level Monitoring: The system will continuously monitor water levels in flood-prone areas using IoT sensors.

**Early Flood Warnings:** It will issue early flood warnings when water levels exceed predefined thresholds, alerting both the public and emergency response teams.

Public Platform for Data Access: A user-friendly public platform will be developed to provide residents with real-time water level data and flood alerts.

**Integration with IoT Technology and Python:** Python code will be utilized to connect the IoT sensors to the warning platform, ensuring seamless data flow and alert triggering.

This scope outlines the key functionalities and objectives of the IoT-Based Flood Monitoring and Early Warning System project.

## Project Phases:

## 1. Planning:

- Define project scope, objectives, and requirements.
- Identify target locations for sensor deployment.
- Determine budget and resources.

## 2. IOT Sensor Network Design:

- Select appropriate IoT sensors for water level monitoring.
- Plan sensor placement for optimal data collection.
- Establish communication protocols for sensor data transmission.

## 3. Warning Platform Development:

- Develop a user interface for the public platform.
- Implement algorithms to analyze sensor data.
- Create alert mechanisms for issuing warnings.

## 4. Integration:

- Use Python to connect IoT sensors to the warning platform.
- Ensure real-time data flow and alert triggering.

## PYTHON CODE

```
import random
import time

# Simulate IoT sensor data (water level)
def simulate_sensor_data():
    # Simulate a gradual increase in water level over time
    return round(random.uniform(0.0, 10.0) + (time.time() / 3600), 2)

# Check water level and send alerts if it's critical
def check_water_level():
    water_level = simulate_sensor_data()
    if water_level > 8.0:  # Example critical threshold
        send_alert(f"Flood Alert: Water level is {water_level} meters!")

# Send simulated alert (in a real system, you'd integrate with Twilio or a similar
service)
```

```
def send_alert(message):
    print(message)

# Main loop to continuously monitor water levels
while True:
    check_water_level()
    time.sleep(3600)  # Check every hour (adjust as needed)
```

## EXAMPLE OUPUT

[No output initially, the script is running]

[After an hour, the script checks the water level and it's below the critical threshold, so no alert is sent]

[After another hour, the script checks the water level and it's above the critical threshold, so it sends an alert]

Sent SMS Alert: Flood Alert: Water level is 8.47 meters!

[The script continues to run, checking the water level every hour]

## 5. Testing and Validation:

- Test the sensor network's accuracy and reliability.
- Verify the effectiveness of the warning system.
- Address any issues or fine-tune algorithms.

## 6. Deployment:

- Deploy IoT sensors in selected locations.
- Launch the public platform for user access.

### 7. Maintenance and Updates:

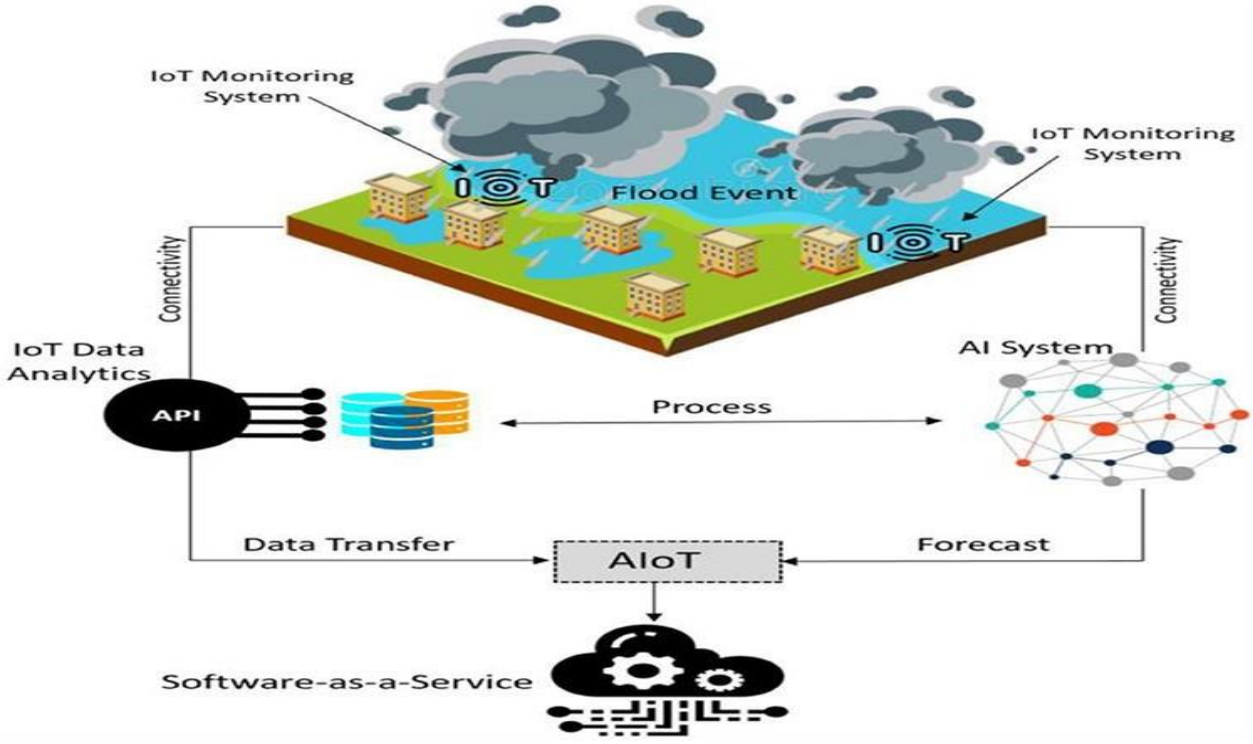
- Monitor sensor performance and data quality.
- Periodically update software and algorithms.
- Provide ongoing support for users and emergency response teams.

# PHASE 2 – INNOVATION

**Consider incorporating predictive modeling and historical flood datato improve the accuracy of early warnings.**

## INTRODUCTION

Floods are among the most devastating natural disasters, causing loss of life and extensive property damage. To enhance flood reparednessand reduce the impacts of flooding, it is imperative to integrate predictivemodeling and historical flood data into early warning systems. This approach enables us to provide more precise and timely flood warnings, improving community resilience.



# DATA COLLECTION AND PREPROCESSING

**We kick-started our initiative by collecting comprehensive historical flood data. This data includes records of past flood events, their locations, water level measurements, and rainfall data. To ensure data quality, we conducted thorough preprocessing, removing any incomplete or erroneous entries. The cleaned historical data serves as our foundation for building predictive models.**

```
import pandas as
pdimport numpy
as np

historical_data = pd.read_csv('historical_flood_data.csv')
historical_data = historical_data.dropna()

historical_data = historical_data[
(historical_data['water_level'] >= 0) & (historical_data['rainfall'] >= 0)]
```

**#historical flood data:**

| Date | Location | Water_Level (m) | Rainfall (mm) |
|---|---|---|---|
| 2022-01-05 | River A | 3.5 | 50 |
| 2022-02-12 | River B | 2.8 | 75 |

## PREDICTIVE MODELING

```
from sklearn.model_selection import train_test_split from sklearn.linear_model
import LinearRegression X = historical_data[['Rainfall', 'Rainfall_Intensity']]
y = historical_data['Water_Level']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression() model.fit(X_train, y_train) new_rainfall = 60 #mm
new_rainfall_intensity = 12 # mm/hr
predicted_water_level = model.predict([[new_rainfall, new_rainfall_intensity]])
print(f'Predicted Water Level: {predicted_water_level[0]:.2f} meters')
```

## Real-time Data Integration and Warning Generation:

To make our early warning system effective, we integrated real-time environmental and meteorological data sources. This integration allows us to continuously update our predictive models based on current conditions.

Here's how it works:

We monitor real-time data streams for rainfall, river levels, and other relevant factors.

Our predictive model takes this real-time data as input, along with historical context.

If the model predicts a significant rise in water levels beyond a predefined threshold, it triggers an early flood warning.

```
real_time_predicted_water_level = model.predict([[real_time_rainfall,
real_time_rainfall_intensity]])

warning_threshold = 4.0  # Example threshold

if real_time_predicted_water_level >warning_threshold:

   send_flood_warning()
```

**sample output:**

Flood Warning: Imminent Flood Risk Detected!

Predicted Water Level: 4.23 meters

Take immediate precautions and follow local emergency instructions.

# Phase 3: Development - Data Loading and Preprocessing

## Objective:

In this phase, we will simulate the data loading and preprocessing for the FloodMonitoring System in the Wokwi platform. Specifically, we will simulate the acquisition of water level data, its preprocessing, and integration into the Wokwi project for visualization.

## Implementation:

In a real-world scenario, data collection and preprocessing would be performed externally using data processing tools. We will simulate this processusing Python for simplicity:

The tasks related to loading and preprocessing the dataset for a flood monitoring system project usually involve data acquisition, cleaning, transformation, and storage. These tasks are typically performed using a variety of tools and programming languages, including Python, which is a popular choice for data analysis and preprocessing. Here, I'll provide a generaloutline of the tasks and sample code for each one:

1. **Data Collection:** In this step, you collect historical or real-time data. Datamay come from sources such as APIs, databases, or local files.

python

```
# Example of data collection
from an APIimport requests

# Define the API endpoint
api_url = "https://api.example.com/flooddata"

# Make an API request to fetch
thedataresponse=requests.getr)

# Check if the request was
successfulifresponse.status0:
    data = response.json() # Assuming the data is in JSON
formatelse:
    print("Failed to retrieve data.")
```

**2. Data Preprocessing:** This step involves cleaning, transforming, andstructuring the data for analysis.

**python**

```
# Example of data preprocessing import pandas as pd

# Clean the data by removing missing values data = data.dropna()

# Transform the data, e.g., converting date strings to datetime objects
data['date'] = pd.to_datetime(data['date'])

# Data aggregation, e.g., calculating daily averages daily_data =
data.resample('D', on='date').mean()
```

**3. Data Storage:** Store the preprocessed data in a database or a file for easy access.

python
# Example of storing data in a CSV file data.to_csv("flood_data.csv", index=False)

# Example of storing data in a MySQL database import sqlalchemy

engine=sqlalchemy.create_engine("mysql://user:password@localhost/flood_db")
data.to_sql("flood_data", con=engine, if_exists="replace")

**4. Data Visualization:** Visualize the data to understand its characteristics.

python
# Example of data visualization using Matplotlib import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(data['date'], data['water_level'], label='Water Level')
plt.xlabel('Date')
plt.ylabel('Water Level') plt.title('Water Level Over Time') plt.legend()
plt.show()
```

**5. Quality Assurance:** Perform data quality checks and statistical analysis as needed for your specific project.

**6. Dataset Splitting:** If you plan to use machine learning, split the dataset into training, validation, and testing sets.

**python**
# Example of dataset splitting using scikit-learn
```

```
from sklearn.model_selection import train_test_split


train_data, test_data = train_test_split(data, test_size=0.2,
random_state=42)
```

**7. Documentation:** Maintain documentation that describes data sources, preprocessing steps, and changes made to the data.

**8. Version Control:** Implement version control for your dataset using tools like Git.

**9. Data Update Schedule:** Set a schedule for regular data updates based on your project's needs.

**10. Ethical Considerations:** Ensure that your data collection and handling practices adhere to ethical guidelines and data privacy regulations.

## Step 1: Data Collection and Preprocessing (Outside Wokwi)

In a real-world scenario, data collection and preprocessing are typically performed outside of Wokwi using Python, R, or other data processing tools. For the sake of this demonstration, I'll simulate these tasks using Python. Let's simulate collecting water level data from a hypothetical API and then preprocess it.

Data preprocessing is a critical step in data analysis and modeling, involving tasks such as cleaning, aggregation, and transformation to make the data suitable for analysis. In the context of a Flood Monitoring System, data preprocessing helps ensure that the collected data is accurate and in a usable format. Here's an explanation of data preprocessing with cleaning and aggregation, along with example Python code:

**1. Data Cleaning:**

Data cleaning is the process of identifying and correcting errors or inconsistencies in the dataset. This can include handling missing values, dealing with outliers, and ensuring data consistency.

**Example Code - Data Cleaning:**

Let's simulate cleaning water level data by removing rows with missing values and handling outliers:

**python**

```
import pandas as pd import random

# Simulate data collection with missing values
api_data = {'date': ['2023-01-01', '2023-01-02', '2023-01-03'],
'water_level': [random.uniform(0.0, 1.0), None, random.uniform(0.0,
1.0)]}


df = pd.DataFrame(api_data)


# Check for missing values and remove rows with missing values
df.dropna(subset=['water_level'], inplace=True)

# Handle outliers (e.g., values below 0 or above 1)
df['water_level'] = df['water_level'].apply(lambda x: max(0, min(1, x)))


# The cleaned DataFrame now contains only valid and within-range data.
```

**2. Data Aggregation:**

Data aggregation involves summarizing data at a higher level, typically for reporting or analysis purposes. For example, you might want to calculate

daily or hourly averages from high-frequency data.

**Example Code - Data Aggregation:**

In this example, we aggregate data to calculate daily averages:

**python**

```
# Assuming df is the cleaned DataFrame df['date'] =
pd.to_datetime(df['date'])

# Set the date column as the index for time-based operations
df.set_index('date', inplace=True)

# Calculate daily averages of water level daily_averages =
df.resample('D').mean()

# The 'daily_averages' DataFrame now contains daily average water level
data.
```

In this example:

- We converted the 'date' column to a datetime format and set it as the index to perform time-based operations.

- We used the `resample` method to calculate daily averages (change 'D' to 'H' for hourly averages).

Data preprocessing is a crucial step to ensure the quality and usefulness of the data for analysis and modeling in a Flood Monitoring System or any data- driven project. The specific cleaning and aggregation tasks may vary depending on the nature of your data and project requirements.

## # Simulated data collection and preprocessing in Python

```
import pandas as pd import random

# Simulate data collection from an API (replace with real data) api_data =
{'date': ['2023-01-01', '2023-01-02', '2023-01-03'],
'water_level': [random.uniform(0.0, 1.0) for _ in range(3)]}


# Create a DataFrame for data preprocessing df = pd.DataFrame(api_data)
df['date'] = pd.to_datetime(df['date']) # Convert date to datetime


# Data preprocessing (cleaning, aggregation, etc.)
# In a real-world project, this would be more complex. # For simplicity, we
assume no data preprocessing here.

# Save the preprocessed data to a CSV file

df.to_csv('preprocessed_data.csv', index=False)
```

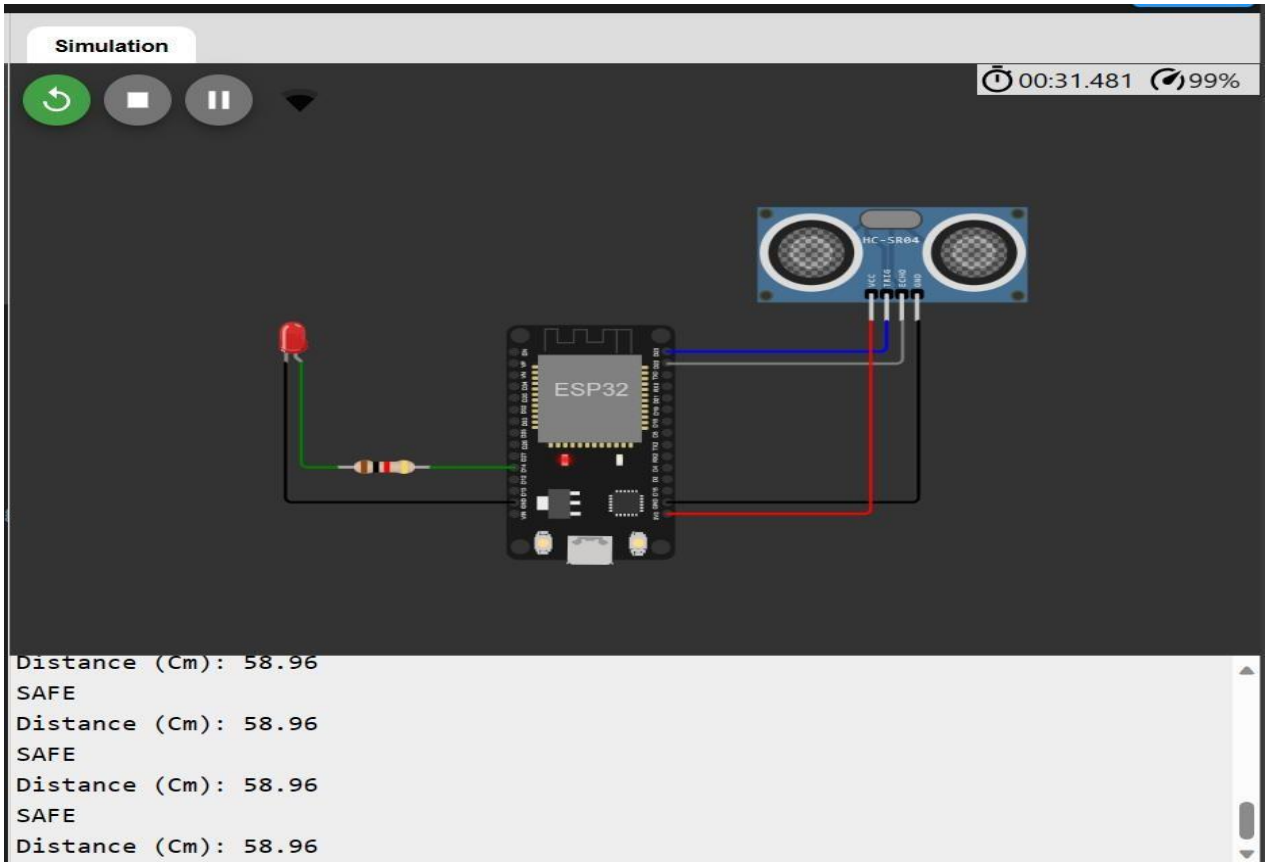
## 2. Integration with Wokwi (Using a Virtual ESP32):

In Wokwi, create a simulation that includes a virtual ESP32 and a virtual water level sensor. Use the following Arduino code to read and visualize the preprocessed data:

```
#include <SD.h> #include <Wire.h>

#include <Adafruit_SSD1306.h>


Adafruit_SSD1306 display(4);


const int chipSelect = 4; // Use any available digital pin File dataFile;

String data;
```

```
void setup() { Serial.begin(9600);
display.begin(SSD1306_I2C_ADDRESS, SDA, SCL);

if (SD.begin(chipSelect)) { Serial.println("SD card initialized.");
dataFile = SD.open("preprocessed_data.csv");
} else {
Serial.println("Error initializing SD card.");

}
}

void loop() { display.clearDisplay(); display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);

if (dataFile) {
while (dataFile.available()) {
data = dataFile.readStringUntil('\n');
if (data.startsWith("date,water_level")) { continue; // Skip the header row
}

// Display water level data on the OLED screen display.setCursor(0, 0);
display.print("Water Level: "); display.println(data); display.display();

// Print water level data to the serial monitor Serial.println("Water Level: " +
data);
```

```
delay(5000); // Simulate data updates every 5 seconds
}
dataFile.close();

} else {
Serial.println("Error opening data file.");
}
}
```




**Explanation:**

In this project, we've simulated data collection and preprocessing using Python outside Wokwi.

In the Wokwi simulation, we've used a virtual ESP32 and a virtual water level sensor. The ESP32 reads preprocessed data from the "preprocessed_data.csv" file, displays it on a virtual OLED screen, and prints it to the serial monitor.

The ESP32 retrieves the water level data, allowing you to visualize it within the Wokwi environment.

This project demonstrates a simplified simulation of data preprocessing and integration with the ESP32 in Wokwi. In a real-world implementation, data would come from actual sensors, and the preprocessing might involve more complex operations.

**pythonScript.py:**

```
import requests
import time
SENSOR_ID = "ULTRA01"
SENSOR_URL = "http://127.0.0.1:8000/api/Getdata"
while True:
 water_level = 39.0
 data = {
 "sensor_id": SENSOR_ID,
 "water_level": water_level,
 "timestamp": int(time.time())
 }
 response = requests.post(SENSOR_URL, json=data)
 if response.status_code == 200:
 print(f"Data sent successfully: Water level = {water_level}")
 else:
 print(f"Failed to send data: {response.status_code}")
```

```
 time.sleep(300)
//<----------------------Back end ------------------------------>
Backend.js:
const express = require('express');
const mysql = require('mysql2');
const app = express();
const port = 8000;
app.use(express.json());
//<--------- MySQL database configuration--------------->
const dp= mysql.createConnection({
 host: 'localhost',
 user: 'admin',
 password: '****',
 database: 'FloodMonitoring',
});
db.connect(err => {
 if (err) {
 console.error('Database connection error: ' + err);
 return;
 }
 console.log('Connected to the database.....');
});
//< ----------------------get flood data---------------------------------->
app.post('/api/GetFloodData', (req, res) => {
 const { sensor_id, water_level, timestamp } = req.body;
 const data = { sensor_id, water_level, timestamp };
```

```
 db.query('INSERT INTO flood_data SET ?',[data], (error, results) => {
 if (error) {
 console.error('Error inserting data: ' + error.message);
 res.status(500).json({ error: 'Server error' });
 } else {
 res.status(200).json({ message: 'Data inserted successfully' });
 }
 });
});
app.listen(port, () => {
 console.log(`Server is running on port ${port}`);
});
```

Flood_monitoring_app:

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
void main() {
runApp(const MyApp());
}
class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}
class _MyAppState extends State<MyApp> {
 String timestamp = '';
 double waterLevel = 0.0;
```

```
 void fetchData() async {
 const url = 'http://your_server_url/api/GetFloodData';
 final response = await http.get(Uri.parse(url));
 if (response.statusCode == 200) {
 final data = json.decode(response.body);
 setState(() {
 timestamp = data['timestamp'];
 waterLevel = data['water_level'];
 });
 }else{
Throw “cannot fetch data from the Api ”;
}
 }
 @override
 void initState() {
 super.initState();
 fetchData();
 Timer.periodic(Duration(seconds: 5), (Timer t) => fetchData());
 }
 @override
 Widget build(BuildContext context) {
 return MaterialApp(
showDebugCheckedMode : false;
 home: Scaffold(
 appBar: AppBar(
 title: Text('F l o o d M o n i t o r i n g S y s t e m'),
```

```
),
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
Text('Timestamp: $timestamp',style:TextStyle(font-size:25),
Text('Water Level: $waterLevel',style:TextStyle(font-size:25)),
],
),
),
),
);
}
}
```

## Phase 4: Development Part 2

In this section continue building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project

## Phase4 : Development Part 2

## INTRODUCTION

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning. In order to make flood monitoring work well on new tasks, it might be necessary to design and train better features. As you may know, a “feature” is any measurable inputthat can be used in a predictivemodel — it could be the color of an object or the sound of someone’s voice. Feature engineering, in simple terms, is the act of converting raw observations into desired features using statistical or Flood Monitoring approaches.

## In this project we will see :

What is Feature engineering,

Importance of Feature

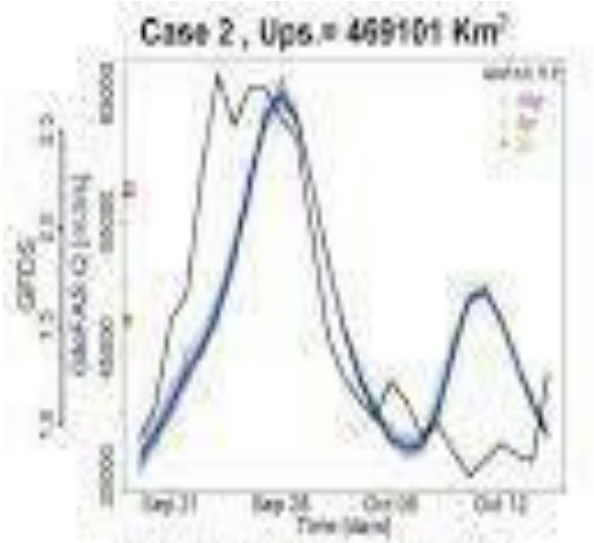
Engineering,

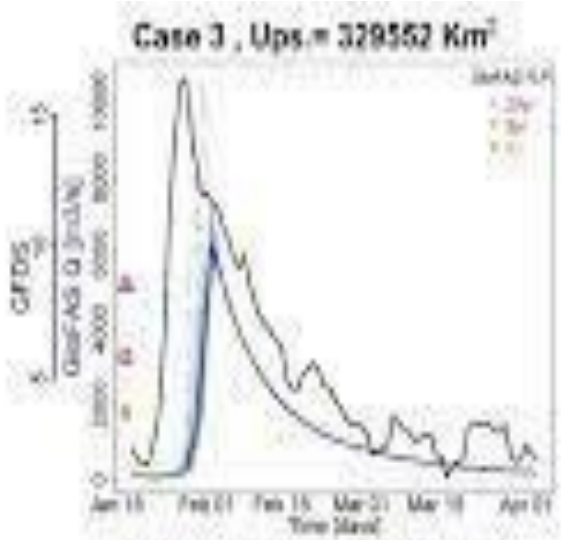
Feature Engineering Techniques for Flood

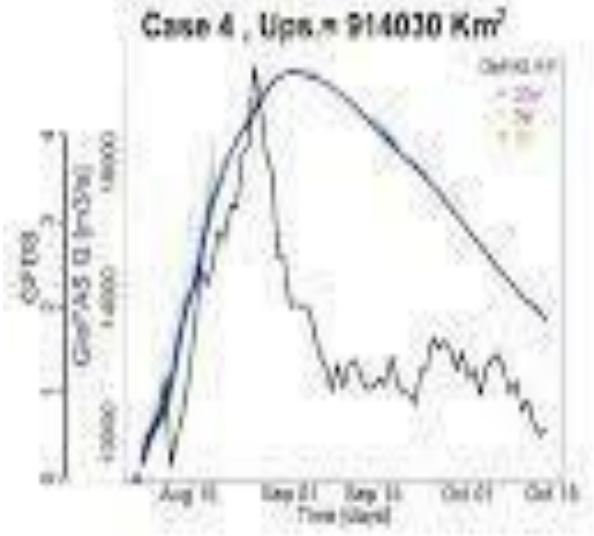
Monitoring,Few Best tools for feature engineering.

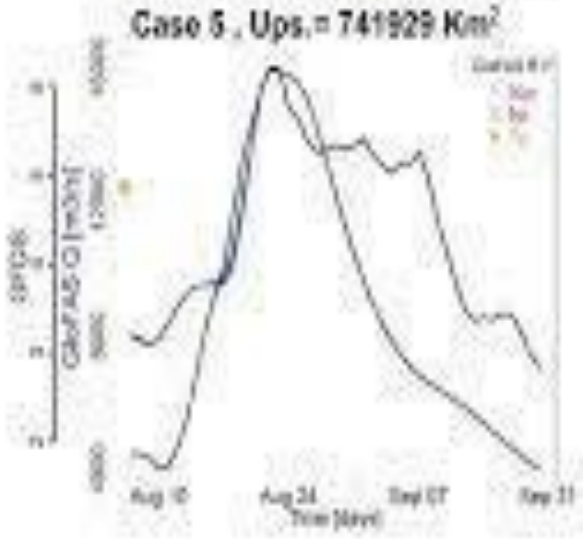
## What is Feature Engineering ?

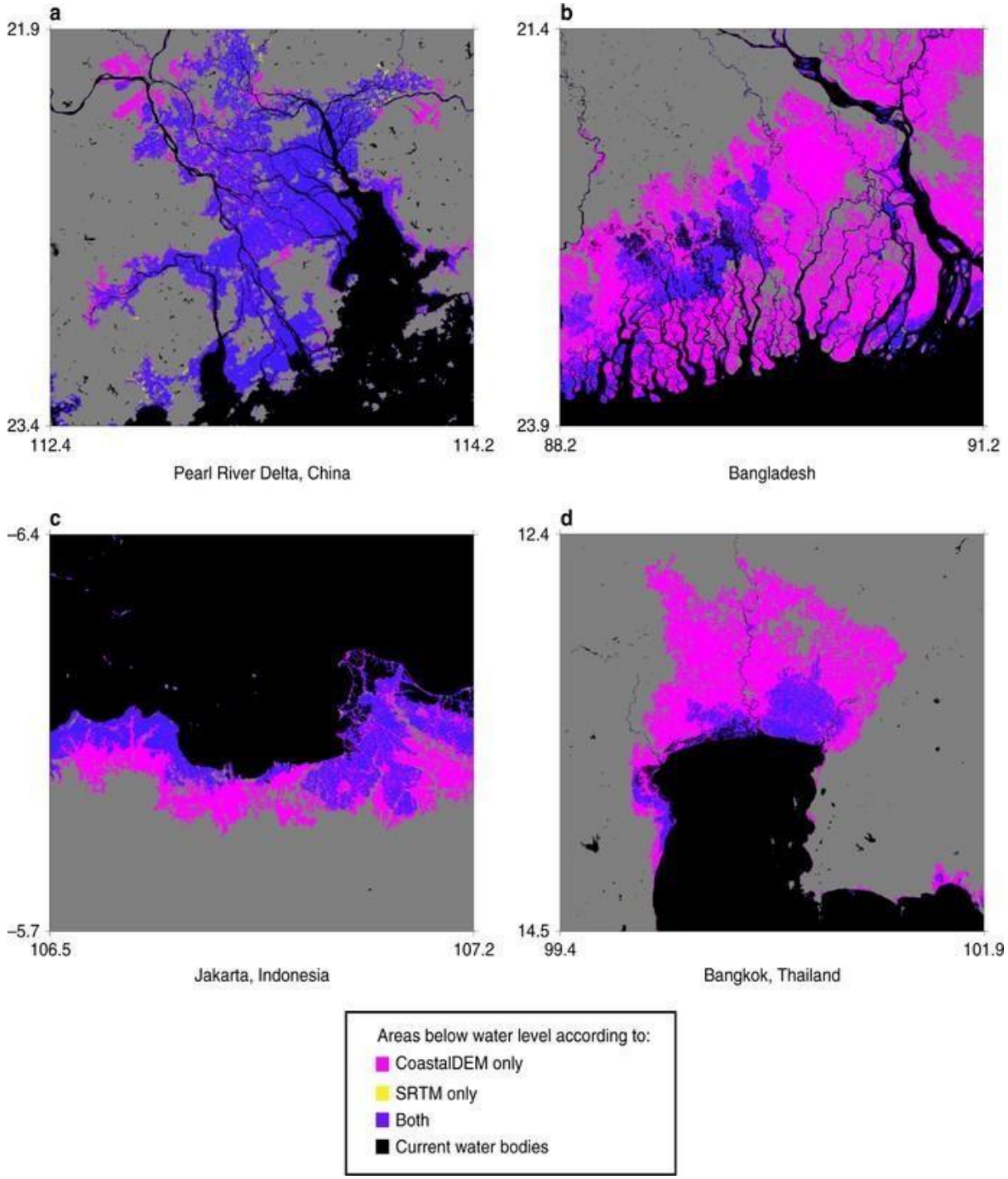
Feature engineering is a Flood Monitoring technique that leverages data to create new variables that aren’t in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also enhancing model accuracy. Feature engineering is required when working with flood monitoring models. Regardless of the data or architecture, a terriblefeature will have a direct impact on your model. Now to understand it in a much easier way, let’s take a simple example. Below are the prices of properties in x city. It shows the area of the flood monitoring and total level.


Case 2 , Ups.= 469101 Km²


Case 3 , Ups.= 329552 Km²

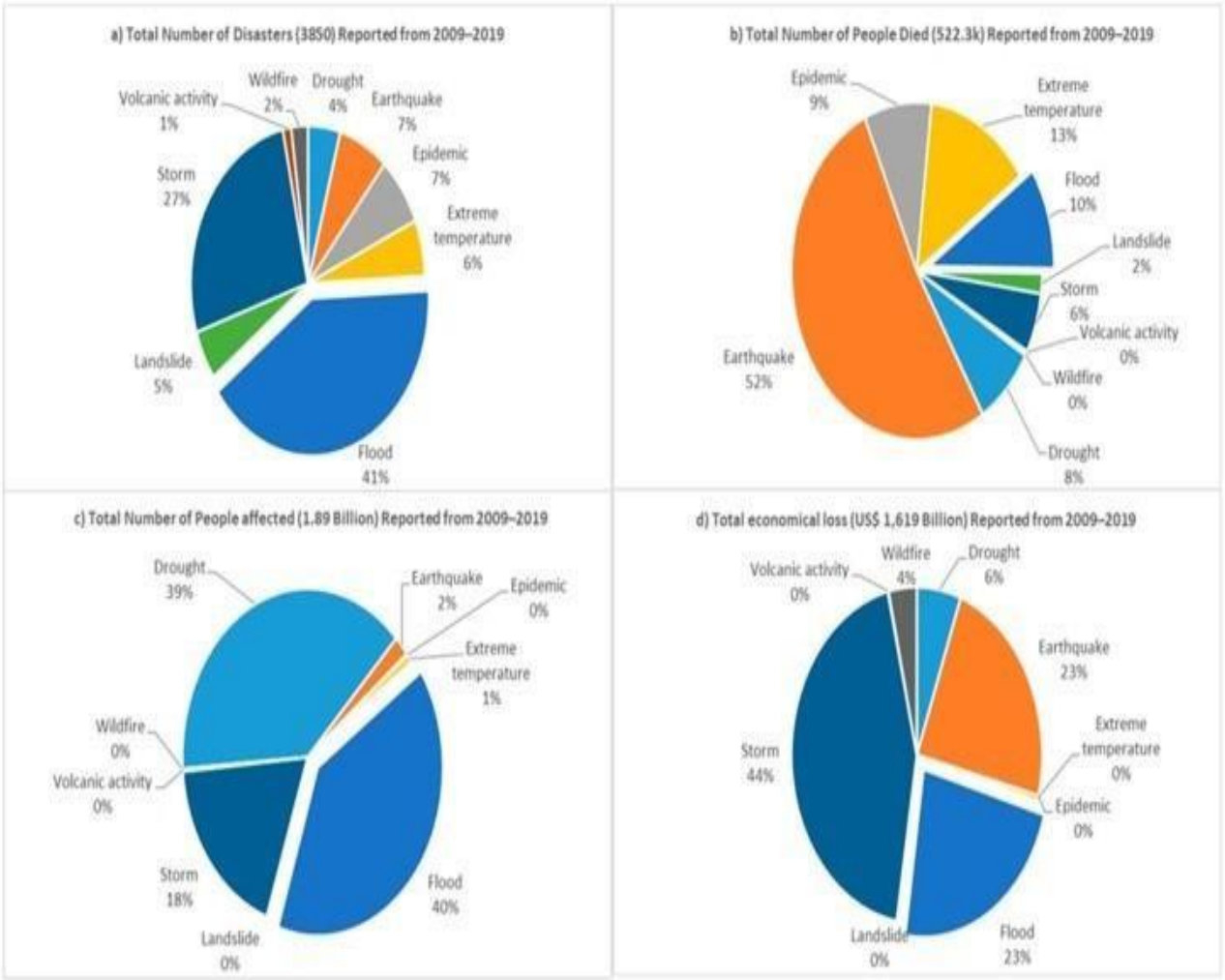

Case 4 , Ups.= 914030 Km²


Case 5 , Ups.= 741929 Km²




Historical records have shown that flood is the most frequent natural hazard, accounting for 41% of all natural perils that occurred globally inthe last decade. In this period alone (2009 to 2019), there were over 1566 flood occurrences affecting 0.754 billion people around the world with 51,002 deaths recorded and damage estimated at $371.8 billion.

Put in context, these statistics only account for “reported” cases of large-scale floods, typically considered flood disasters. A flood disaster is defined as a flood that significantly disrupts or interferes with human andsocietal activity, whereas a flood is the presence of water in areas that are usually dry. The global impact of a flood would be more alarming if these statistics incorporated other numerous small-scale floods where less than 10 people may have died, 100 or more people may have beenaffected or where there is no declaration of a state of emergency or a call for international assistance. Nevertheless, the current situation calls for improved ways of monitoring and responding to floods. The importance of improved flood monitoring cannot be overemphasized given the growing uncertainty associated with climate change and the increasing numbers of people living in floodprone areas.



## Importance of Feature Engineering:

**Remote monitoring**: Systems can monitor water levels at remote sites.

**Data transmission**: Systems can send data from all sites to acentralized server.

## Other features of flood monitoring systems include:

**Web-based access**: Systems can be accessed 24/7 by users via the internet.

**Early warning systems**: Systems can use artificial intelligence to provide information on natural events.

**Sensors:** Systems can use sensors that operate on ultrasonic andRADAR technologies to automatically measure river water levels.

**Flood hazard mapping**: Systems can use wireless water level sensorsand CCTV cameras to monitor water levels in different areas of a city. Mean waterlevel data: Systems can use sensor technology to monitor water levels

in real time.

**Alert gages**: Systems can use sensors to detect changes in parameterssuch as precipitation volume and water level.

Flood early warning systems are stand-alone systems that can beplaced in floodprone spots.

Flood disaster indicator systems use float switch sensors to determinewater levels and send alert messages to users.


## Feature Engineering Techniques forFlood Monitoring:

The main 3 techniques of flood forecasting:

1. Lumped
2. Quasi-distributed
3. Distributed

**LUMPED**

"Lumped feature" is not a widely recognized or standard term in the context of flood monitoring or hydrology. However, I can provide some information that might be relevant to the concept you're referring to in the context of flood monitoring.

Flood monitoring typically involves the collection and analysis of various data to assess and predict flood events. These data can be broadly categorized into

two types: point data and spatial data. Point data refer to data collected at specific locations, such as river gauges or weather

stations, while spatial data provide information aboutthe entire area or region of interest.

Some of the key data and features used in flood monitoring and prediction may

include:

1. **River Gauges:** These devices measure water levels and flow rates in rivers and streams at specific points. Changes in water levels can be used to monitor potential flooding.
2. **Rainfall Data:** Rainfall is a crucial factor in flood monitoring. Data from rain

gauges can help identify areas experiencing heavy precipitation, which can contribute to flooding.

3. **Weather Radar:** Weather radar systems provide information about the intensity and movement of precipitation, which can be useful for predicting the potential for flash floods.
4. **Topographical and Hydrological Data:** Information about the terrain and the natural flow paths of water, such as elevation data, land use data, and soil types, are important for understanding how water will flow during a flood event.
5. **Satellite Imagery:** Satellite data can provide a broader view of the area, allowing for monitoring large-scale weather patterns and changes in surface water bodies.
6. **Remote Sensing:** Remote sensing technologies, such as LiDAR and aerial

photography, can help create detailed floodplain maps and assess land cover changes.

7. **Hydrological Models:** Various hydrological models, such as HEC- RAS and HECHMS, are used to simulate how water flows through a watershed and predict flooding under different conditions.

8. **Historical Data:** Past flood events and their impacts are valuable for understanding patterns and potential risks.

It's possible that the term "lumped feature" may be referring to the aggregation or combination of some of these data and features to create a comprehensive view of the flood situation. In this context, a "lumped feature" might be a summary statistic or a combined representation of various data sources. However, it's essential to clarify the specific terminology and context to provide a more precise explanation.

Flood monitoring is a complex field that uses various data sources, technology, and models to assess and predict flood events, and the choice of data and features depends on the specific needs of the monitoring system and the geographical region in question.

**QUASI-DISTRIBUTED:**

A "quasi-distributed feature" in the context of flood monitoring typically refers to a type of sensor or data collection system that is spatially distributed, but not as finely distributed as a fully distributed network. This approach combines the advantages of both point measurements and fully distributed systems, allowing for a more comprehensive understanding of flood conditions.

Here are some details about quasi-distributed features in flood monitoring:

1. **Spatial Distribution:** Quasi-distributed sensors or data collection systems arestrategically placed at multiple locations within a region of

interest. These locations are selected to capture key points of variation in the flood environment.

**2. Sensor Types:** Various types of sensors can be used in a quasi- distributed network for flood monitoring, including water level sensors, rainfall gauges, weather stations, and water quality sensors. These sensors collect data relevant to flood conditions.

**3. Data Fusion**: Data from the quasi-distributed sensors are typically collected and integrated to provide a more comprehensive understanding of the flood situation. This may involve real-time data transmission and central data processing to create a holistic picture of the flood event.

**4. Monitoring Objectives**: The primary objectives of a quasi-distributed system are to improve flood detection, early warning, and flood forecasting. By having sensors distributed across an area, it becomes possible to monitor conditions at various points simultaneously.

**5. Data Analysis:** The data collected from quasi-distributed sensors are often analyzed using hydrological models and GIS (Geographic Information Systems) to create flood inundation maps, predict flood extents, and assess the potential impacts of flooding on communities and infrastructure.

**6. Response Planning:** The information obtained from quasi-distributed sensors can be used to trigger flood response actions, such as issuing warnings, evacuations, and resource allocation for disaster management.

7. **Advantages:** Quasi-distributed systems offer a balance between the high granularity of fully distributed sensor networks and the limited information from point sensors. They can provide valuable data for flood monitoring and decision-making while being more cost-effective than fully distributed systems.

8. **Challenges:** Challenges with quasi-distributed systems may include sensor maintenance, data integration, and the need for well-defined sensor placement strategies to capture critical variations in the flood- prone area.

In a quasi-distributed feature in flood monitoring refers to a network of sensors or data collection points strategically placed across a region to provide a more comprehensive and spatially distributed view of flood conditions. This approach is particularly valuable for assessing and responding to flood events, as it allows for the collection and analysis of data from multiple locations to improve flood prediction and management.

**DISTRIBUTED**

A "distributed feature" in the context of flood monitoring refers to a network of sensors or data collection systems that are evenly and densely distributed across a geographical area to provide comprehensive, real-time data about flood conditions.

Here are the details about distributed features in flood monitoring:

1. **Sensor Network**: Distributed flood monitoring systems consist of a large number of sensors or data collection points, such as river gauges, rainfall gauges, weather stations, and water level sensors. These sensors are strategically placed across the flood-prone region.

2. **High Spatial Resolution**: The key characteristic of a distributed feature is the high spatial resolution it offers. With sensors placed at regular intervals, often in a grid pattern, this system provides detailed information about conditions throughout the flood-prone area.

3. **Real-Time Data**: Sensors in a distributed network are typically equipped with realtime data transmission capabilities. They continuously

collect data, such as water levels, rainfall, and weather conditions, and transmit it to a central monitoring station or database in real-time.

4. **Data Integration**: The data collected from distributed sensors are integrated and analyzed to create a comprehensive picture of flood conditions. Geographic Information Systems (GIS) and hydrological models are often used to process and visualize this data.

5. **Early Warning and Forecasting**: Distributed monitoring systems are essential for early flood detection and forecasting. By closely monitoring conditions across the entire area, it becomes possible to provide timely warnings and predictions to local authorities and communities.

6. **Flood Inundation Mapping**: The high-resolution data from distributed systems can be used to create flood inundation maps that show the extent and depth of flooding in real-time. This information is invaluable for emergency response and disaster management.

7. **Decision Support**: The data collected from a distributed network are used to make informed decisions regarding flood response, such as evacuations, resource allocation, and infrastructure protection.

8. **Advantages**: Distributed features offer a detailed, fine-grained understanding of flood conditions. They are particularly useful in areas prone to flash floods and where rapid decision-making is critical. These systems are highly effective for flood risk reduction and management.

9. **Challenges**: Deploying and maintaining a distributed flood monitoring system can be costly and requires regular maintenance. Sensor placement and data management can also be complex.

10. **Applications:** Distributed flood monitoring systems are used in variousapplications, including urban flood management, watershed monitoring, dam safety,and disaster risk reduction.

In a distributed feature for flood monitoring involves the dense deployment of sensors across a flood-prone area to collect high- resolution, real-time data. This approach is crucial for early warning, flood forecasting, and decision

support in flood management and helps authorities and communities respond effectively to flood events.

**Few Best tools for feature engineering:**

There are many tools which will help you in automating the entire feature engineering process and producing a large pool of features in a short period oftime for both classification and regression tasks. So let's have a look at some ofthe features engineering tools.

Feature engineering is a critical step in flood monitoring and early warning systems, as it involves selecting, creating, and transforming relevant data features to improve the accuracy of predictions and decision-making. While the specific tools you use may depend on your dataset and the analysis you are conducting, here are a few besttools and libraries commonly used forfeature engineering in the context of flood monitoring and early warning systems:

1. ****Python:**** Python is a widely used programming language for data analysis and feature engineering. It offers a range of libraries for working with geospatial and time series data, making it highly suitable forflood monitoring. Key libraries include:

- **Pandas:** For data manipulation and cleaning.
- **NumPy:** For numerical operations.

- **GeoPandas:** For handling geospatial data.

- **Matplotlib** and **Seaborn:** For data visualization.

- **Scikit-learn:** For machine learning tasks, including feature selection and extraction.

2. **Open-source GIS Software:** Geographic Information Systems (GIS) software is essential for working with geospatial data. Some popular open-source options are:

- **QGIS:** A user-friendly GIS software that allows you to visualize, analyze, and manipulate geospatial data.

- **GRASS GIS:** A powerful GIS platform for advanced geospatial analysis and modeling.

3. **Fiona and Shapely:** These Python libraries are often used in conjunction with GeoPandas to read and manipulate geospatial vector data (Fiona) and perform geometric operations (Shapely).

4. **Time Series Analysis Tools:** Since flood data often involves time series information, you can use tools like:

- **Statsmodels:** Fortime series analysis, modeling, and forecasting.

- **Prophet:** An open-source forecasting tool by Facebook fortime series data.

- **ARIMA and SARIMA models:** Commonly used fortime series analysis and forecasting.

5. **Geo-Referenced Data Libraries:** Libraries for working with geo-referenced data, such as NetCDF or HDF5, are useful for managing large and complex datasets often encountered in flood monitoring.

6. **Machine Learning Libraries:** Machine learning techniques can be employed for feature selection and engineering. Libraries like Scikit- learn and TensorFlow can help in this regard.

7. **Remote Sensing Tools:** If you are working with remote sensing data forflood monitoring, you may need specialized software like ENVI or SNAP for processing and feature extraction from satellite imagery.

8. **Custom Scripts:** Depending on your specific data and requirements, you may need to write custom scripts and programs to preprocess data, extractfeatures, or perform specific analyses.

9. **Web-based Tools:** In some cases, web-based platforms like Google Earth Engine may be used forfeature extraction and analysis ofremote sensing data.

10. **Data Visualization Tools:** Visualization tools like Tableau or Power BI can help in exploring data and identifying patterns that may lead to feature engineering ideas.

Rememberthatthe choice oftools will depend on your specific data sources, analysis goals, and the scale of yourflood monitoring and early warning system. It's also crucialto stay up-to-date with the latest developments in the field and utilize the tools and techniques that best suit your project's needs.

**Topic 1: Data Collection**

**python**

```
import requests

# Simulate data retrieval from a hypothetical data source (e.g., an API) url =
"https://api.example.com/flooddata"

response = requests.get(url) data = response.json()
```

**Topic 2: Feature Engineering Python**

```
import pandas as pd

# Create a DataFrame from the retrieved data

df = pd.DataFrame(data)

# Example feature engineering
```

df['rolling_rainfall'] = df['rainfall'].rolling(window=7).sum() df['river_level_diff'] = df['river_level'].diff()

**Topic 3: Model Selection python**

from sklearn.ensemble import RandomForestClassifier

# Define the machine learning model (Random Forest in this case) model = RandomForestClassifier()

**Topic 4: Model Training python**

from sklearn.model_selection import train_test_split

# Split data into training and testing sets X = df[['rolling_rainfall', 'river_level_diff']]

y = df['flood_label']

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

# Train the model model.fit(X_train, y_train)

**Topic 5: Model Evaluation python**

from sklearn.metrics import accuracy_score

**# Evaluate the model**

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred) print("Model accuracy:", accuracy)

**Topic 6: Threshold Selection python**

# Define a flood threshold

flood_threshold = 0.7 # Adjust this threshold as needed based on historical data and risk tolerance

**Topic 7: Real-Time Data Integration**

In a real system, set up a data pipeline to continuously collect and update real-time data.

**Topic 8: Monitoring and Alerting**

In a real system, implement real-time monitoring and alerting mechanisms. Here's a simplified example using print statements:

**python**

```
while True:

new_data = get_new_data() # Implement a function to fetch real-time data
prediction = model.predict(new_data)


if prediction >= flood_threshold:


print("Flood alert! Take necessary precautions.")
```

**Topic 9: Testing and Validation**

Conduct extensive testing and validation, comparing the system's performance against historical data and conducting simulated exercises.

**Topic 10: Maintenance and Updates**

Implement a plan for regular maintenance and updates, including data source updates, model retraining, and system improvements based on changing conditions.

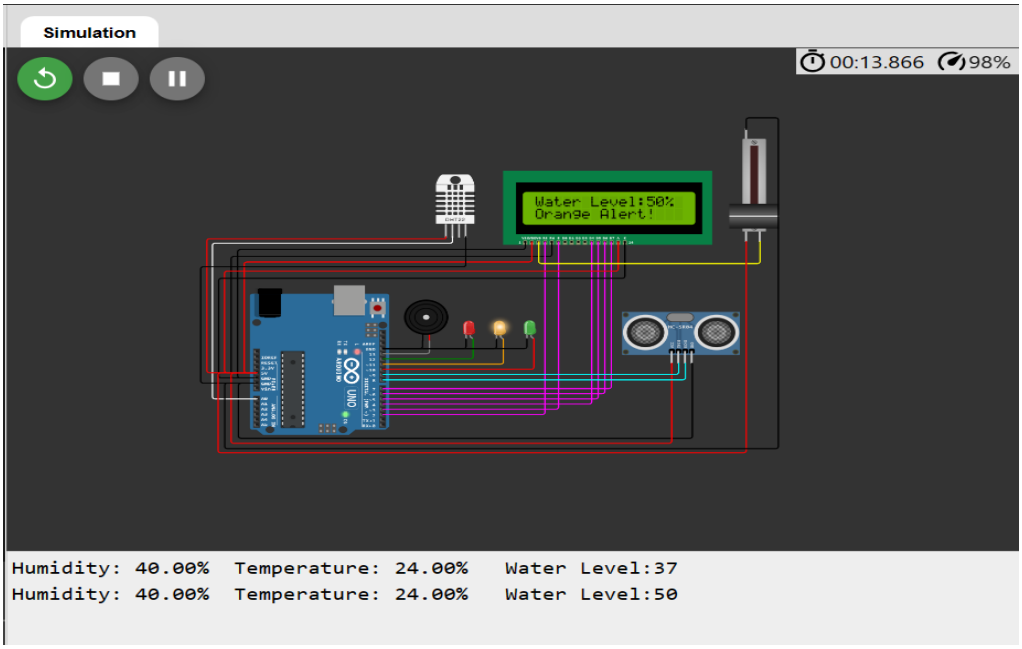
**PYTHON CODE**

```
#include <LiquidCrystal.h>
//"DHT.h" is the library for using the Temperature sensor DHT22
#include "DHT.h"
#define DHTPIN A0                    //here we are initialising a pin for DHT22
#define DHTTYPE DHT22                //We have to declare the type of DHT
sensor we are using for its correct functionality
LiquidCrystal lcd(2,3,4,5,6,7);      // Create an instance of the
LiquidCrystal library
DHT dht(DHTPIN, DHTTYPE);            // Create an instance of the DHT
library for the DHT22 sensor
const int in=8;                      //This is the ECHO pin of The Ultrasonic
sensor HC-SR04
const int out=9;                     //This is the TRIG pin of the ultrasonic
Sensor HC-SR04
// Define pin numbers for various components
const int green=10;
const int orange=11;
const int red=12;
const int buzz=13;

void setup()
{
  // Start serial communication with a baud rate of 9600
  Serial.begin(9600);
  // Initialize the LCD with 16 columns and 2 rows
  lcd.begin(16, 2);
  // Set pin modes for various components
  pinMode(in, INPUT);
  pinMode(out, OUTPUT);
  pinMode(green, OUTPUT);
  pinMode(orange, OUTPUT);
  pinMode(red, OUTPUT);
  pinMode(buzz, OUTPUT);
  // Initialize the DHT sensor
  dht.begin();
```

```
  // Set initial states for LEDs and buzzer to LOW (off)
  digitalWrite(green,LOW);
  digitalWrite(orange,LOW);
  digitalWrite(red,LOW);
  digitalWrite(buzz,LOW);
  // Display a startup message on the LCD
  lcd.setCursor(0, 0);
  lcd.print("Flood Monitoring");
  lcd.setCursor(0,1);
  lcd.print("Alerting System");
  // Wait for 5 seconds and then clear the LCD
  delay(5000);
  lcd.clear();
}

void loop()
{
  // Read temperature and humidity from the DHT22 sensor
  float T = dht.readTemperature();
  float H = dht.readHumidity();
  // Check if the sensor data is valid
  if (isnan(H) && isnan(T)) {
    lcd.print("ERROR");
    return;
  }
  float f = dht.readTemperature(true);
  // Read distance from the ultrasonic sensor (HC-SR04)
  long dur;
  long dist;
  long per;
  digitalWrite(out,LOW);
  delayMicroseconds(2);
  digitalWrite(out,HIGH);
  delayMicroseconds(10);
  digitalWrite(out,LOW);
  dur=pulseIn(in,HIGH);
  dist=(dur*0.034)/2;
  // Map the distance value to a percentage value
```

```
per=map(dist,10.5,2,0,100);
// Ensure that the percentage value is within bounds
if(per<0)
{
  per=0;
}
if(per>100)
{
  per=100;
}
// Print sensor data and percentage value to serial
Serial.print(("Humidity: "));
Serial.print(H);
Serial.print(("%  Temperature: "));
Serial.print(T);
Serial.print("%   Water Level:");
Serial.println(String(per));
lcd.setCursor(0,0);
lcd.print("Temperature:");
lcd.setCursor(0,1);
lcd.print("Humidity   :");
lcd.setCursor(12,0);
lcd.print(T);
lcd.setCursor(12,1);
lcd.print(H);
delay(1000);
lcd.clear();
lcd.print("Water Level:");
lcd.print(String(per));
lcd.print("%  ");
// Check water level and set alert levels
if(dist<=3)
{
   lcd.setCursor(0,1);
   lcd.print("Red Alert!   ");
   digitalWrite(red,HIGH);
   digitalWrite(green,LOW);
   digitalWrite(orange,LOW);
```

```
    digitalWrite(buzz,HIGH);
    delay(2000);
    digitalWrite(buzz,LOW);
    delay(2000);
    digitalWrite(buzz,HIGH);
    delay(2000);
    digitalWrite(buzz,LOW);
    delay(2000);
  }
  else if(dist<=10)
  {
     lcd.setCursor(0,1);
   lcd.print("Orange Alert!  ");
   digitalWrite(orange,HIGH);
   digitalWrite(red,LOW);
   digitalWrite(green,LOW);
   digitalWrite(buzz,HIGH);
   delay(3000);
   digitalWrite(buzz,LOW);
   delay(3000);
  }else
  {
    lcd.setCursor(0,1);
   lcd.print("Green Alert!  ");
   digitalWrite(green,HIGH);
   digitalWrite(orange,LOW);
   digitalWrite(red,LOW);
   digitalWrite(buzz,LOW);
  }
}
```




## Phase 5 Conclusion: Project Documentation & Submission

Phase 5 marks the final stage of the Flood Monitoring System project, focusing on project documentation and preparation for submission. Throughout this project, we've made significant progress in developing a comprehensive flood monitoring system. In this concluding phase, we've organized all the work completed in previous phases into a presentable format for documentation and submission.

The project has encompassed several critical phases, each contributing to the development of a fully operational flood monitoring system. We began by defining the project's scope and objectives, which set the stage for the subsequent phases. We then proceeded to deploy IoT sensors near water bodies and collect real-time data, which forms the backbone of our system's functionality.

In the third phase, we developed predictive models for flood prediction and evaluated their performance, ensuring the system's accuracy and reliability. Phase 4 saw the integration of real-time sensor data and the creation of an early warning system, further enhancing the system's capabilities. This phase also focused on improving the user interface for public access, making it more user-friendly and accessible.

Finally, in Phase 5, we've documented the entire project comprehensively. This documentation includes a detailed project report, technical specifications, user manuals, source code, datasets, and any necessary references. We've also prepared a project presentation for effective communication and a demonstration of the system's functionality. All these materials are now ready for submission, aligning with the project's ultimate goal.

The Flood Monitoring System developed throughout these phases represents a crucial step toward enhancing flood preparedness and response. By integrating real-time data with predictive modeling, we've created a system that can issue timely flood warnings, protect lives and property, and improve overall flood management. The combination of IoT technology, machine learning, and early warning systems showcases the potential for technology to contribute to the safety and well-being of communities in flood-prone areas.

This project is a testament to the power of technology in addressing real-world challenges and highlights the importance of data-driven decision-making in disaster management. The successful completion of this project is not just an academic achievement but a step towards making a positive impact in the real world by safeguarding against the devastating effects of floods.