Rising Waters: A Machine Learning Approach to Flood Prediction

Team Members :-

- Pallela Shanmukha Srinadh
- Koteswarararao Vulli
- Shaik Abdul Vahid
- Karthik Sai Vyasarouthu

Team ID :-

- LTVIP2026TMIDS56565

Project Overview:

Flood Prediction using Machine Learning is a vital application that aims to forecast and predict flood occurrences with high accuracy. By analyzing historical weather data, river levels, terrain information, and other relevant factors using machine learning algorithms, this project helps in early warning and mitigation of potential flood events. The goal is to provide timely alerts and actionable insights to authorities, communities, and individuals to minimize the impact of floods on lives and infrastructure.

Scenario 1: Early Warning Systems

One of the primary use cases is the development of early warning systems for flood-prone areas. By analyzing real-time data and predicting flood risks, authorities can issue timely alerts to residents, enabling them to take preventive measures such as evacuation or reinforcement of flood defenses.

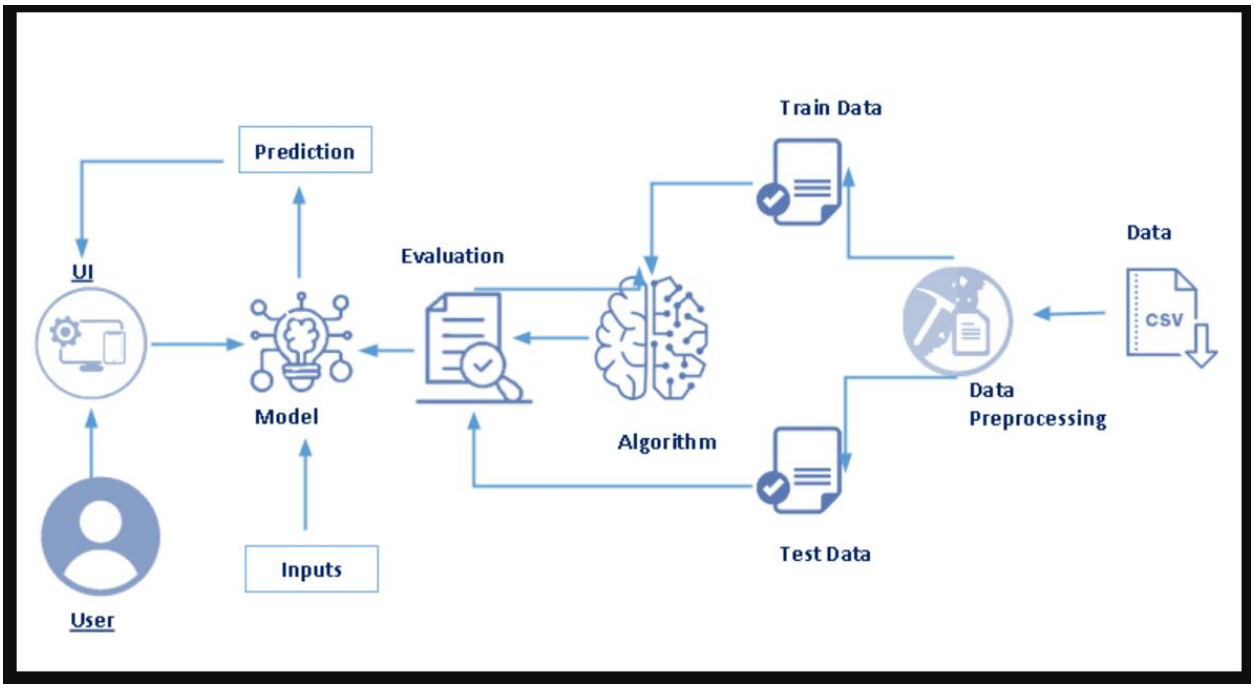
Scenario 2: Disaster Response Planning

Flood prediction plays a crucial role in disaster response planning. Emergency services can use the predictions to allocate resources, plan rescue operations, and coordinate relief efforts effectively, reducing the response time and maximizing the impact of assistance during flood emergencies.

Scenario 3: Infrastructure Resilience

City planners and engineers can leverage flood predictions to design resilient infrastructure. By incorporating flood risk assessments into urban development projects, they can implement measures such as flood barriers, drainage systems, and green infrastructure to mitigate flood damage and protect critical infrastructure assets.

Technical Architecture:



Pre-Requisites:

To complete this project, you must require the following software, concepts, and packages

VS Code Installer:

- Refer to the link to download the VS Code navigator
- https://www.youtube.com/watch?v=mIVB-SNycKI

Python packages:

- Open anaconda prompt as administrator
- Type **"pip install numpy"** and click enter.
- 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 click enter.
- Type **"pip install seaborn"** and click enter.
- Type **"pip install Flask"** and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- **ML Concepts**

Supervised learning:

- https://www.youtube.com/watch?v=QeKshry8pWQ&t=3s

Unsupervised learning:

- https://www.youtube.com/watch?v=D6gtZrsYi6c

Metrics:

- https://www.youtube.com/watch?v=aWAnNHXIKww

Flask:

- https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

Write what are all the technical aspects that students would get if they complete this project.

1. Knowledge of Machine Learning Algorithms.

2. Knowledge of Python Language with Machine Learning

3. You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.

4. You will be able to know how to pre-process/clean the data using different data pre-processing techniques.

5. Applying different algorithms according to the dataset and based on visualization.

6. Real-Time Analysis of Project

7. Building ease of User Interface (UI)

8. Navigation of ideas towards other projects(creativeness)

9. Knowledge of building ML models.

10. How to build web applications using the Flask framework.

Project Flow:

1. Install Required Libraries.

2. Data Collection.
   - Collect the dataset or Create the dataset

3. Data Preprocessing.

- Import the Libraries.
- Importing the dataset.
- Understanding Data Type and Summary of features.
- Take care of missing data
- Data Visualization.
- Drop the column from DataFrame & replace the missing value.
- Splitting the Dataset into Dependent and Independent variables
- Splitting Data into Train and Test.

4. Model Building

- Training and testing the model
- Evaluation of Model
- Saving the Model

5. Application Building

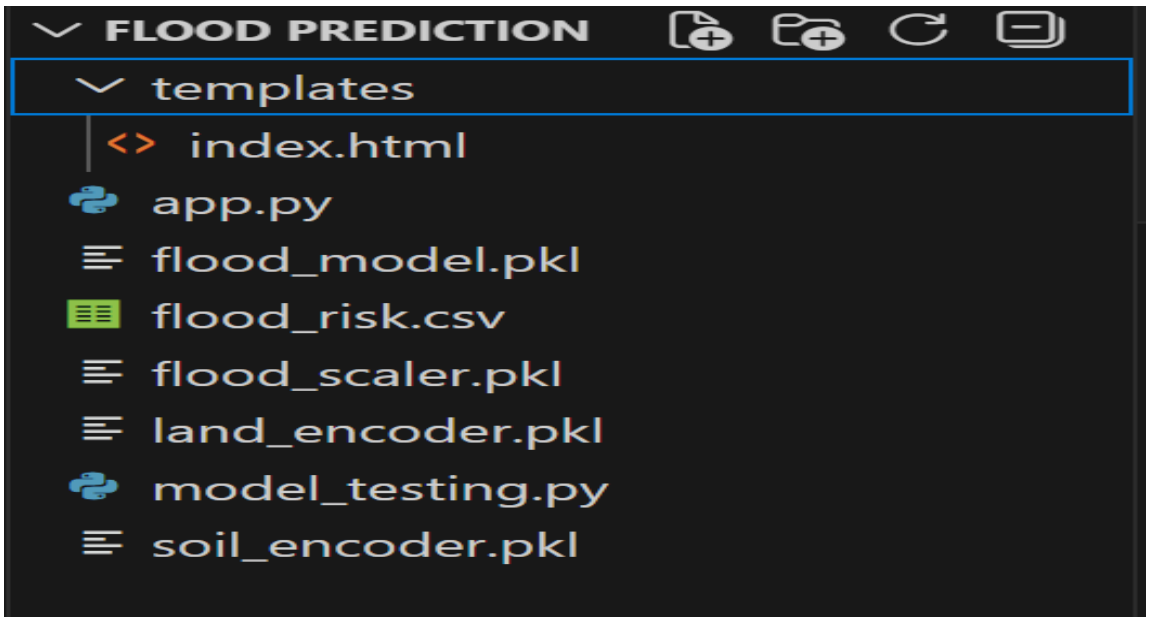
- Create an HTML file
- Build a Python Code

6. Final UI

- Dashboard of the flask app

Project Structure:

Create a Project folder that contains files as shown below



- We are building a Flask Application that needs HTML page **"index.html"** stored in the templates folder and a python script **app.py** for server-side scripting
- The model is built in the VS Code **model_testing.py**
- The dataset is saved as flood_risk.csv
- **.pkl** files used are saved models used in app.py
- The templates mainly used are **"index.html"** for showcasing the UI
- The flask app is denoted as **app.py**

Data Collection:

ML depends heavily on data, without data, it is impossible for an "AI" to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

Download the data set

Download the dataset from the below link.
https://www.kaggle.com/datasets/s3programmer/flood-risk-in-india

Visualizing and analyzing the data:

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing the libraries:

Import the necessary libraries as shown in the image

Import the required libraries for the model to run. The first step is usually importing the libraries that will be needed in the program.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import joblib  # Used to save the model for the web app
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Reading the Dataset:

- Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.
- In pandas, we have a function called read_csv() to read the dataset. As a parameter, we have to give the directory of CSV file.

```
# --- STEP 1: LOAD AND PREPROCESS ---
df = pd.read_csv('flood_risk.csv')
```

Descriptive Analysis:

To check the first five rows of the dataset, we have a function called **head( )**.

```
print(df.head())
```

```
    Latitude  Longitude  Rainfall (mm)  ...  Infrastructure  Historical Floods  Flood Occurred
0  18.861663  78.835584     364.999155  ...               1                  0               1
1  35.570715  77.654451      92.255998  ...               0                  1               0
2  29.227824  73.108463     173.319847  ...               1                  1               1
3  25.361096  85.610733     331.640318  ...               1                  1               1
4  12.524541  81.822101     241.044672  ...               1                  0               0

[5 rows x 14 columns]
```

Understanding Data Type and Summary of features

- How the information is stored in a DataFrame or Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data: numeric and text data types.
- Numeric data types include integers and floats.
- Text data type is known as Strings in Python, or Objects in Pandas. Strings can contain numbers and / or characters.
- For example, a string might be a word, a sentence, or several sentences.
- Will see how our dataset is, by using **info()** method.
- **info()** method provides the summary of dataset.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Latitude                10000 non-null  float64
 1   Longitude               10000 non-null  float64
 2   Rainfall (mm)           10000 non-null  float64
 3   Temperature (C)         10000 non-null  float64
 4   Humidity (%)            10000 non-null  float64
 5   River Discharge (m3/s)  10000 non-null  float64
 6   Water Level (m)         10000 non-null  float64
 7   Elevation (m)           10000 non-null  float64
 8   Land Cover              10000 non-null  object
 9   Soil Type               10000 non-null  object
 10  Population Density      10000 non-null  float64
 11  Infrastructure          10000 non-null  int64
 12  Historical Floods       10000 non-null  int64
 13  Flood Occurred          10000 non-null  int64
dtypes: float64(9), int64(3), object(2)
memory usage: 1.1+ MB
None
```

- As you can see in our dataset there is no textual data, all the set of data is in float and integer type.
- **Describe ()** functions are used to compute values like count, mean, standard deviation give a summary type of data.

```
print(df.describe().T)
```

	count	mean	std	min	25%	50%	75%	max
Latitude	10000.0	22.330627	8.341274	8.000337	15.143537	22.283330	29.460184	36.991813
Longitude	10000.0	82.631366	8.389542	68.004575	75.364428	82.671007	89.937897	96.997820
Rainfall (mm)	10000.0	250.025197	143.386879	0.024062	126.873956	251.034047	372.336927	499.950489
Temperature (C)	10000.0	29.961401	8.669838	15.000166	22.405717	30.000907	37.413488	44.993681
Humidity (%)	10000.0	59.749104	23.142734	20.001339	39.541778	59.497375	80.038163	99.997772
River Discharge (m3/s)	10000.0	2515.722946	1441.706442	0.042161	1284.782376	2530.451944	3767.229862	4999.698480
Water Level (m)	10000.0	7.526822	4.314869	0.004052	3.808271	7.563142	11.287038	14.995348
Elevation (m)	10000.0	2496.122387	1429.840315	0.650056	1259.992034	2496.157188	3739.110005	4999.375496
Population Density	10000.0	5021.468442	2882.591520	2.289000	2491.766601	5074.392879	7474.228752	9999.169530
Infrastructure	10000.0	0.502000	0.500021	0.000000	0.000000	1.000000	1.000000	1.000000
Historical Floods	10000.0	0.498700	0.500023	0.000000	0.000000	0.000000	1.000000	1.000000
Flood Occurred	10000.0	0.500000	0.500025	0.000000	0.000000	0.500000	1.000000	1.000000

Data Pre-processing:

As we have understood how the data is. Let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

? Handling missing values
? Handling categorical data
? Handling outliers
? Splitting the dependent and independent variables
? Splitting dataset into training and test set
? Feature scaling

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

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.

```
print(df.isnull().any())
```

```
Latitude                  False
Longitude                 False
Rainfall (mm)             False
Temperature (C)           False
Humidity (%)              False
River Discharge (m3/s)    False
Water Level (m)           False
Elevation (m)             False
Land Cover                False
Soil Type                 False
Population Density        False
Infrastructure            False
Historical Floods         False
Flood Occurred            False
dtype: bool
```

As you can see that, our data do not contain any null values. Here the function is null.any() return the Boolean values False. When that return True, which means that particular in dataset has missing values. So we can skip this step.

Handling outliers:

The dataset features exhibit consistent value ranges with no significant extreme anomalies; therefore, manual outlier treatment was not required.

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.

```
# FIX: Separate LabelEncoders to prevent mapping collision
# This ensures Land Cover categories don't overwrite Soil Type categories
le_land = LabelEncoder()
df['Land Cover'] = le_land.fit_transform(df['Land Cover'])

le_soil = LabelEncoder()
df['Soil Type'] = le_soil.fit_transform(df['Soil Type'])
```

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:

```
# Feature Selection (Independent Variables X, Dependent Variable y)
X = df.drop('Flood Occurred', axis=1)
y = df['Flood Occurred']
```

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 into 80% Training and 20% Testing
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Feature scaling:

Standard Scaler: **Sklearn** its main scaler, the **StandardScaler**, uses a strict definition of standardization to standardize data.

```
# Scaling the independent variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Model Building:

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved

based on its performance. To evaluate the performance confusion matrix and classification report is used.

Decision tree model:

A function named **DecisionTree** is created and train and test data are passed as the parameters.

Random forest model:

A function named **RandomForest** is created and train and test data are passed as the parameters.

KNN model:

A function named **KNN** is created and train and test data are passed as the parameters.

XGBoost model:

A function named **XGBoost** is created and train and test data are passed as the parameters.

```
# Import the models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
```

```
# --- STEP 2: DEFINE AND TEST MODELS ---
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "XGBoost": XGBClassifier(eval_metric='logloss', random_state=42)
}
```

Compare the model:

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

```python
results = {}

print("--- Accuracy Comparison ---")
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    results[name] = acc
    print(f"{name}: {acc*100:.2f}%")
```

After calling the function, the results of models are displayed as output.

```
--- Accuracy Comparison ---
Decision Tree: 84.70%
Random Forest: 88.75%
KNN: 84.15%
XGBoost: 88.95%
```

Evaluating performance of the model:

Random Forest was selected for the final deployment because it provided high accuracy (88.75%) with greater stability and less risk of overfitting compared to boosting methods. Its ensemble approach ensures reliable predictions across the diverse environmental features present in our dataset.

```python
# We use Random Forest as the production model
best_model = models["Random Forest"]
y_pred_best = best_model.predict(X_test)
```

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.

```
# Save all assets so the Web App can use them
joblib.dump(best_model, 'flood_model.pkl')
joblib.dump(scaler, 'flood_scaler.pkl')
joblib.dump(le_land, 'land_encoder.pkl')
joblib.dump(le_soil, 'soil_encoder.pkl')

print("\n✅ SUCCESS: Model, Scaler, and Encoders saved as .pkl files!")
```

Build Flask Application:

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 serverside 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 “flood_model.pkl”. 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 (flood_model.pkl).
- Templates folder which contains index.HTML file.

The user interface was developed using HTML5 for structure and CSS3 for custom styling. To ensure a responsive and modern design, the Bootstrap 5 framework was integrated, utilizing its grid system and utility classes. Additionally, Font Awesome icons were used to improve the visual hierarchy and user experience of the input form.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Rising Waters | Flood Prediction</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body { background-color: #f0f2f5; }
        .card { border-radius: 15px; border: none; box-shadow: 0 10px 30px rgba(0,0,0,0.1); }
        .header { background: #0d6efd; color: white; padding: 25px; border-radius: 15px 15px 0 0; }
    </style>
</head>
<body>
    <div class="container py-5">
        <div class="row justify-content-center">
            <div class="col-lg-8 col-md-10 card p-0">
                <div class="header text-center">
                    <h2 class="fw-bold">Flood Risk Assessment System</h2>
                    <p class="mb-0">Environmental Data Analysis</p>
                </div>

                <form action="/predict" method="post" class="p-4">
                    <div class="row g-3">
                        <div class="col-md-4">
                            <label class="form-label">Rainfall (mm)</label>
                            <input type="number" step="any" name="rainfall" class="form-control" value="{{ inputs.rainfall }}" required>
                        </div>
                        <div class="col-md-4">
                            <label class="form-label">Temp (C)</label>
                            <input type="number" step="any" name="temp" class="form-control" value="{{ inputs.temp }}" required>
                        </div>
                        <div class="col-md-4">
                            <label class="form-label">Humidity (%)</label>
                            <input type="number" step="any" name="humidity" class="form-control" value="{{ inputs.humidity }}" required>
                        </div>

                        <div class="col-md-6">
                            <label class="form-label">River Discharge (m3/s)</label>
                            <input type="number" step="any" name="discharge" class="form-control" value="{{ inputs.discharge }}" required>
                        </div>
                        <div class="col-md-6">
                            <label class="form-label">Water Level (m)</label>
                            <input type="number" step="any" name="water_level" class="form-control" value="{{ inputs.water_level }}" required>
                        </div>
```

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:

To build the backend, we import the necessary Flask modules for routing and template rendering. We also use the **joblib** library (a modern alternative to Pickle) to load the serialized machine learning model and its associated preprocessing files.

```python
from flask import Flask, render_template, request
import joblib
import numpy as np
```

Initializing the App and Loading Models:

An object of the Flask class is created as our WSGI application, using the __name__ argument to determine the root path. We then load the four essential assets generated during the training phase: The **Random Forest model**, the **StandardScaler**, and the two **LabelEncoders**.

```
app = Flask(__name__)

# Load your existing model (trained on all 13 features)
model = joblib.load('flood_model.pkl')
scaler = joblib.load('flood_scaler.pkl')
le_land = joblib.load('land_encoder.pkl')
le_soil = joblib.load('soil_encoder.pkl')
```

Routing to the HTML Page:

We use the **@app.route('/')** decorator to bind the root URL to the **home()** function. When a user accesses the web server, the **render_template** function is called to display the **index.html** interface.

```
@app.route('/')
def home():
    return render_template('index.html',inputs=None)
```

Handling Predictions:

The **predict()** function is responsible for the core logic of the application. It utilizes the **POST** method to receive data from the user interface, processes it, and returns the result.

- **Data Extraction:** The form data is captured as a dictionary.
- **Categorical Encoding:** Text inputs for "**Land Cover**" and "**Soil Type**" are transformed into numerical values using the loaded encoders.
- **Feature Alignment:** Since the model was trained on 13 features, the function constructs a full feature vector by combining the user's inputs with default "neutral" values for the remaining variables.
- **Scaling and Prediction:** The StandardScaler normalizes the data before the **model.predict()** function generates the final result.

```python
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get data from the form
        data = request.form.to_dict()

        # 1. Transform categorical inputs
        land_val = le_land.transform([data['land_cover']])[0]
        soil_val = le_soil.transform([data['soil_type']])[0]

        # 2. Neutral values for removed fields
        default_lat = 20.0
        default_long = 78.0
        default_pop_dense = 3000
        default_infra = 0

        # 3. Construct full 13-feature list
        full_features = [
            default_lat, default_long,
            float(data['rainfall']), float(data['temp']),
            float(data['humidity']), float(data['discharge']),
            float(data['water_level']), float(data['elevation']),
            land_val, soil_val,
            default_pop_dense, default_infra,
            int(data['history'])
        ]

        # 4. Predict
        scaled_features = scaler.transform([full_features])
        prediction = model.predict(scaled_features)

        result = "⚠️ DANGER: High Flood Risk!" if prediction[0] == 1 else "✅ SAFE: Low Flood Risk"
        bg = "danger" if prediction[0] == 1 else "success"

        # IMPORTANT: We pass 'inputs=data' back to the HTML
        return render_template('index.html',
                               prediction_text=result,
                               status=bg,
                               inputs=data)

    except Exception as e:
        return render_template('index.html', prediction_text=f"Error: {str(e)}", status="warning")
```

Main Function:

To execute the application, the main block is called. We run the app with **debug=True** during the development phase to allow for real-time code updates and error tracking.

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

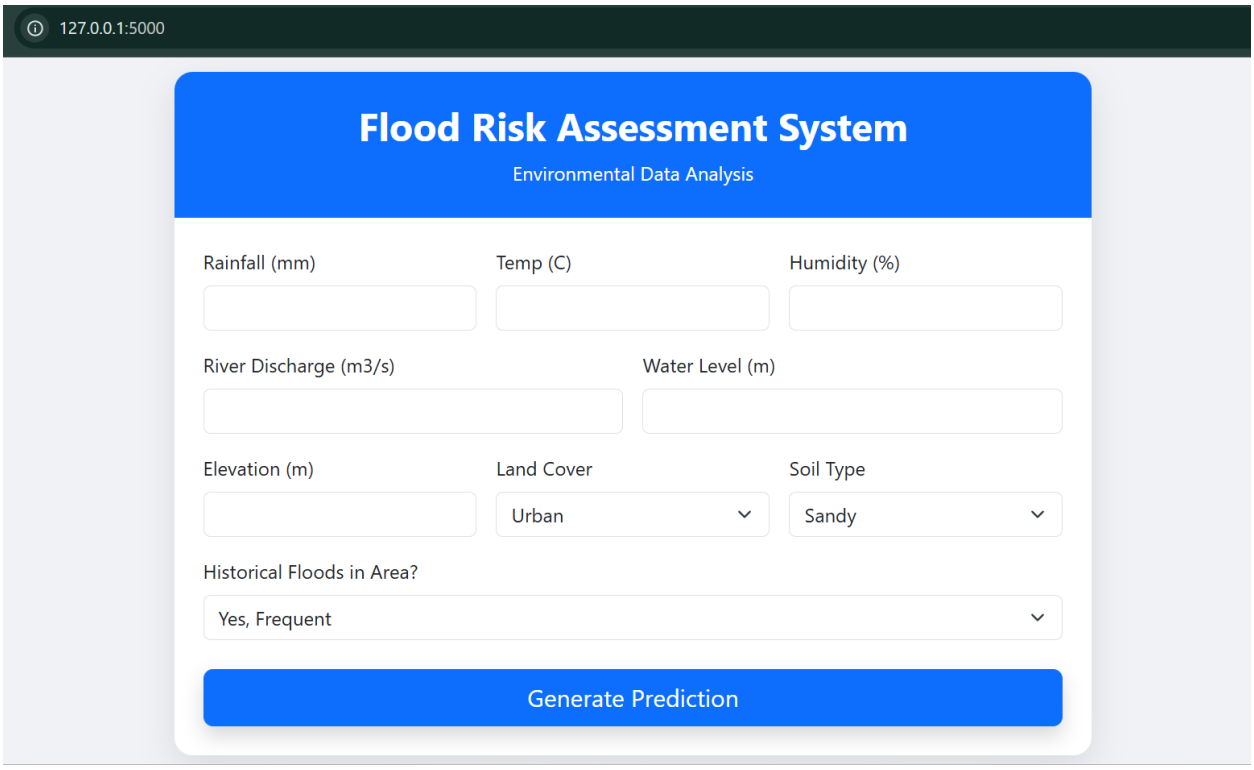
Run the application:

- Open VS Code from start menu.
- Navigate to the folder where your app.py resides.
- Now type the “python app.py” command in the command terminal.
- It will show the local host where your app is running on http://127.0.0.1.5000/
- Copy that local host URL and open that URL in browser. It does navigate you to the where you can view your web page.
- Enter the values, click on predict button and see the result/predict on web page.

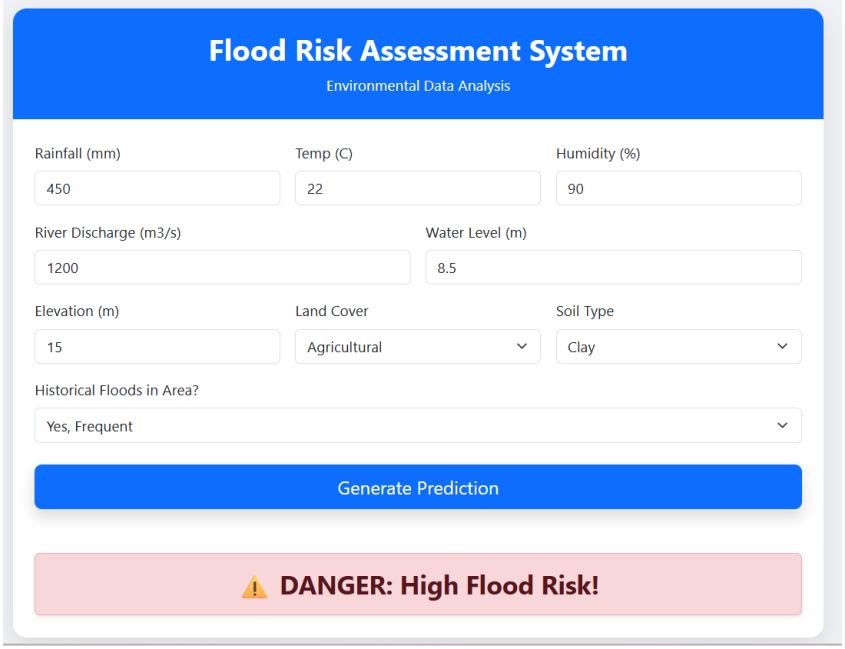
```
python -u "c:\Users\HP\Desktop\Flood Prediction\app.py"
```

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 363-503-306
127.0.0.1 - - [21/Feb/2026 15:21:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Feb/2026 15:21:22] "GET /favicon.ico HTTP/1.1" 404 -
```

Output:



Output-1:



Output-2: