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

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:

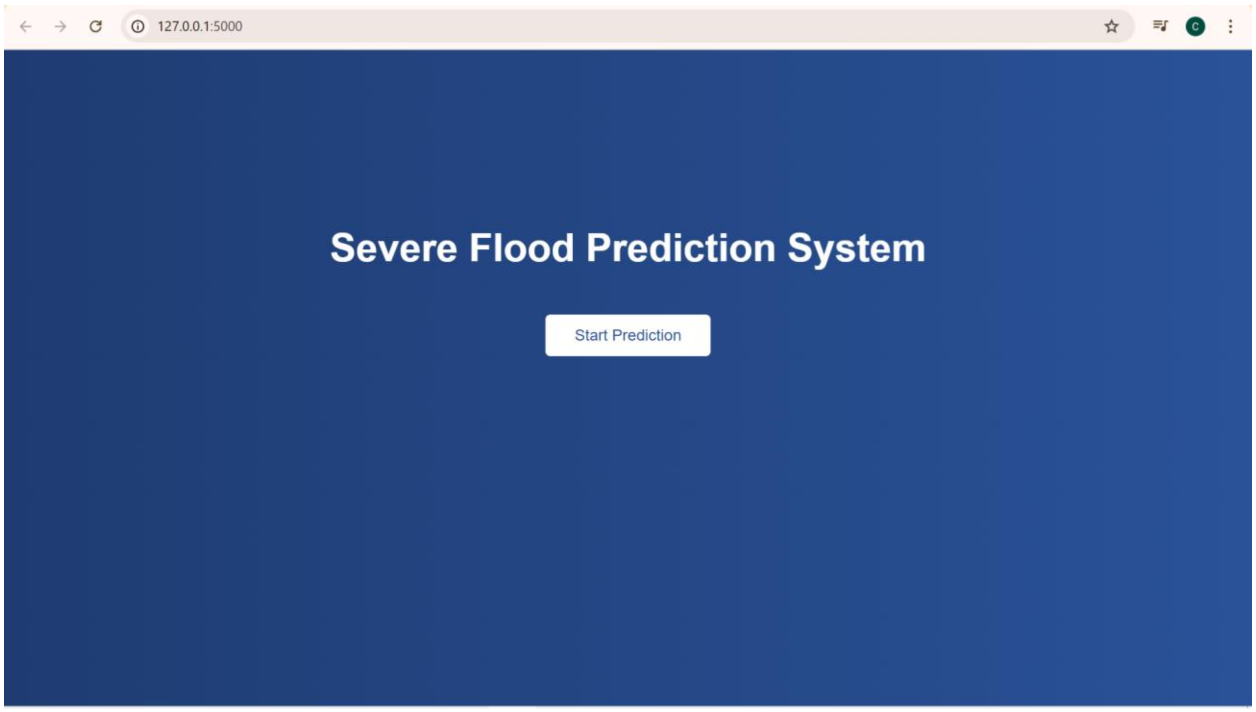
- ➢ Flask Frame Work with Machine Learning Model In this section we will be building a web application which is integrated to the model we built. An UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased 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.

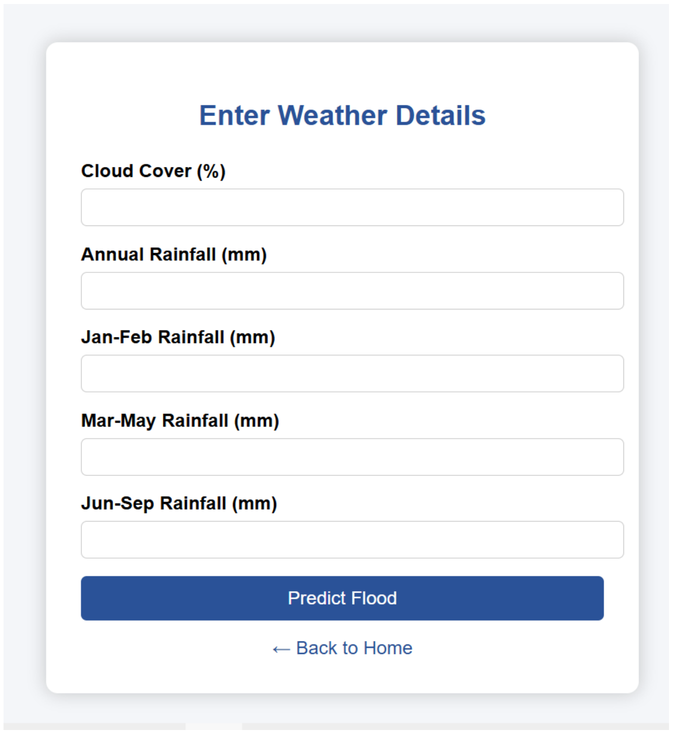
Here, we created 4 html pages- home.html, image.html, imageprediction.html, intro.html.

- 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

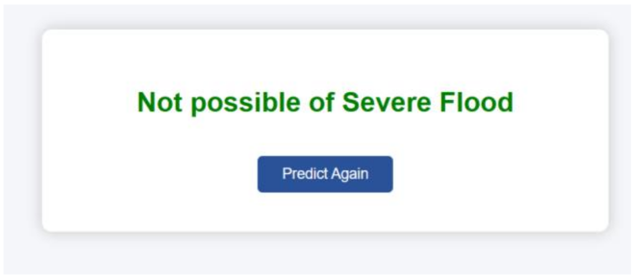


- ➢ This page collects user input
- ➢ Takes 5 independent variables:
    - Cloud Cover
    - Annual Rainfall
    - Jan-Feb Rainfall
    - Mar-May Rainfall
    - Jun-Sep Rainfall



Displays prediction result
✓ Shows:

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



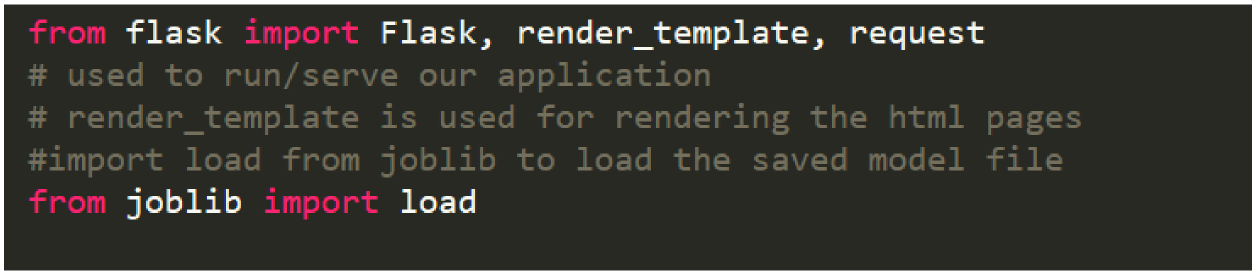
Build python code:

- 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.

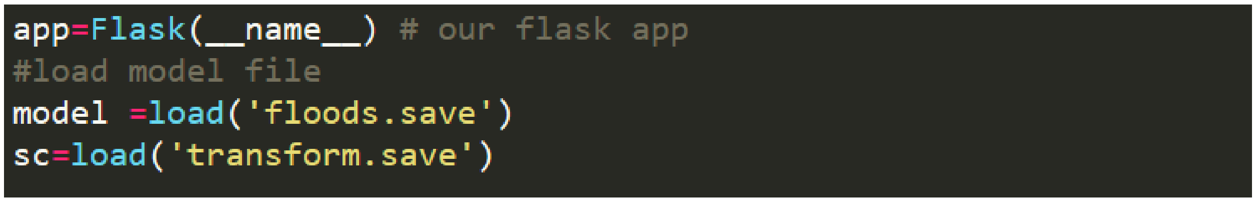
Importing Libraries

- 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

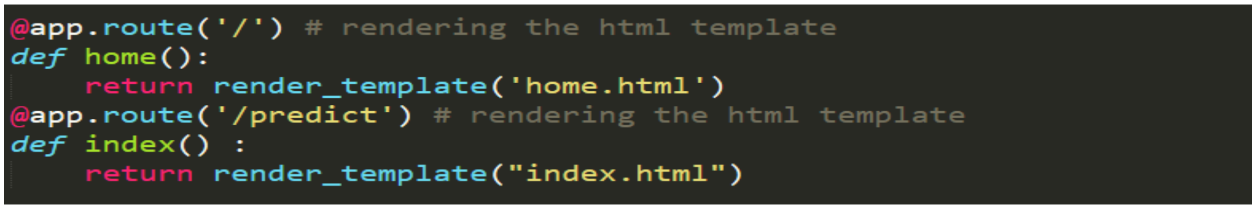


```
app=Flask(__name__) # our flask app
#load model file
model =load('floods.save')
sc=load('transform.save')
```

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.



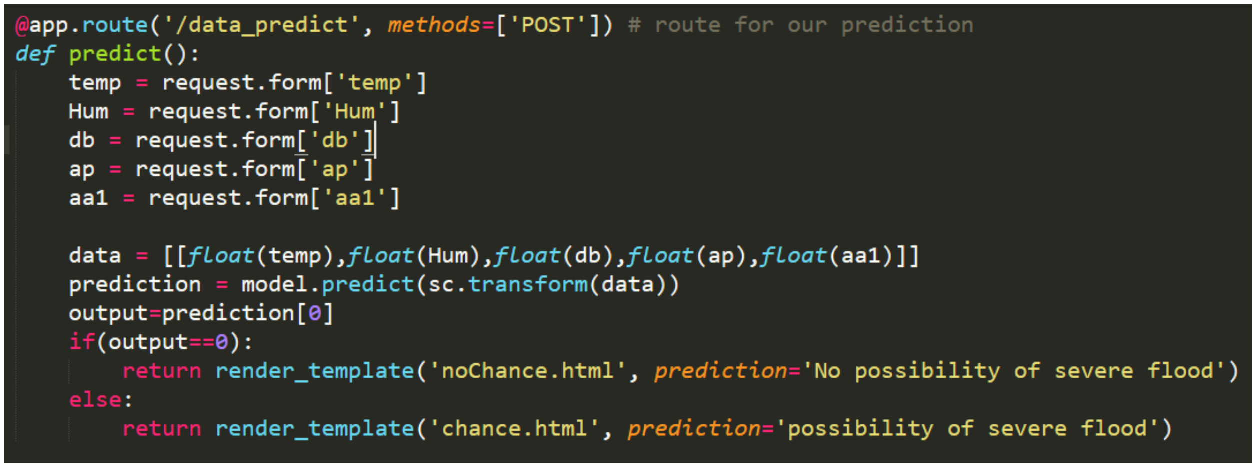
```python
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")
```

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

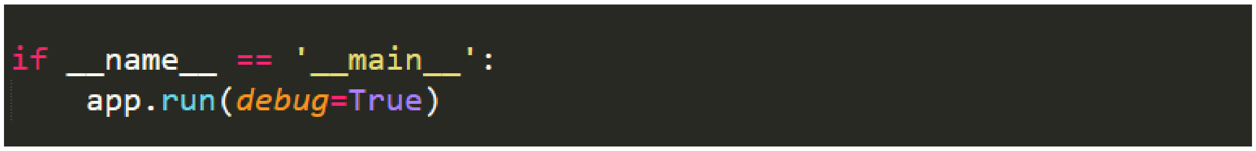
- predict() – is taking the values from the prediction page and storing it into a variable and then we are creating a DataFrame along with the values and 5 independent features and finally, we are predicting the values using or loaded model which we build and storing the output in a variable and returning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



```python
@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



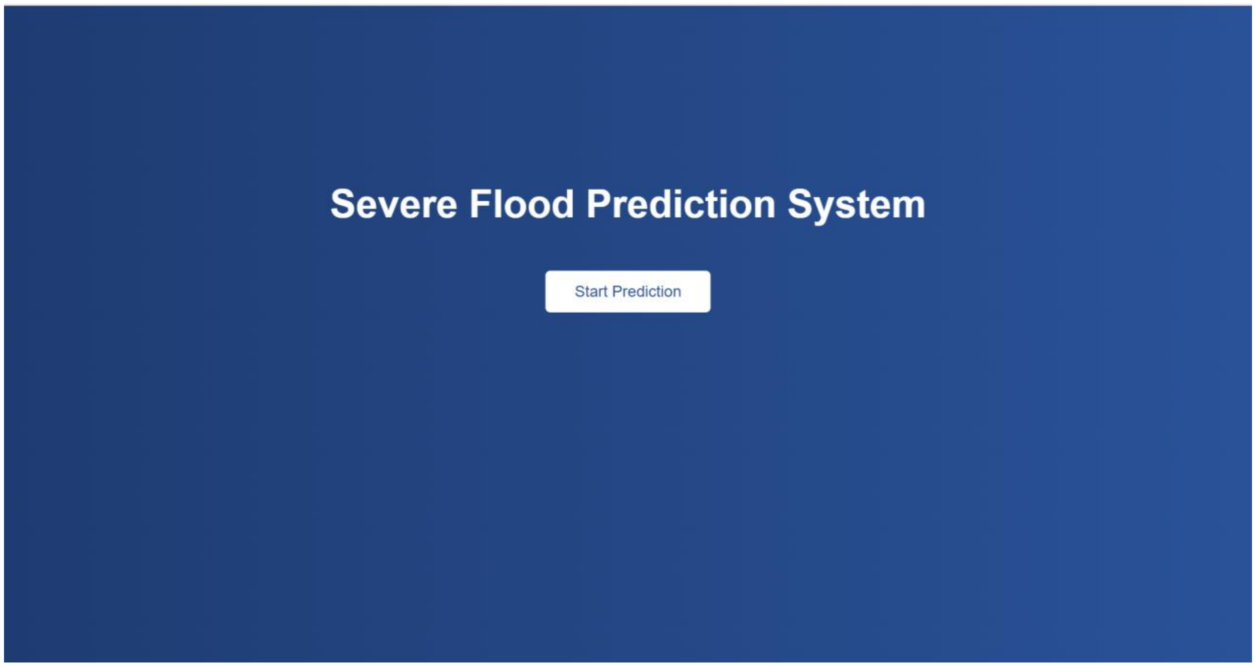
```
if __name__ == '__main__':
    app.run(debug=True)
```

FULL PYTHON CODE:

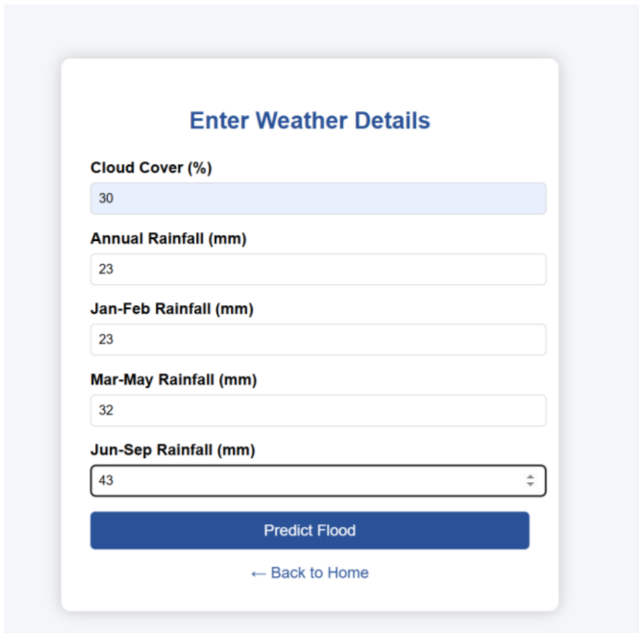
```
from flask import Flask, render_template, request

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



Enter Weather Details

Cloud Cover (%): 30

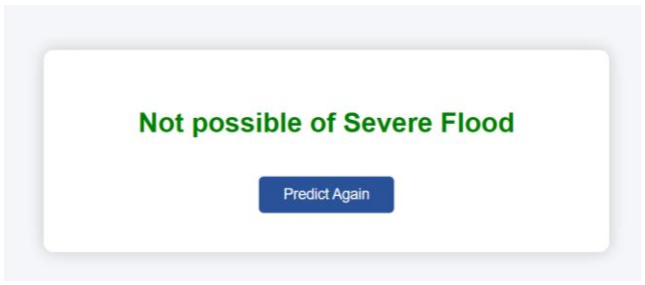
Annual Rainfall (mm): 23

Jan-Feb Rainfall (mm): 23

Mar-May Rainfall (mm): 32

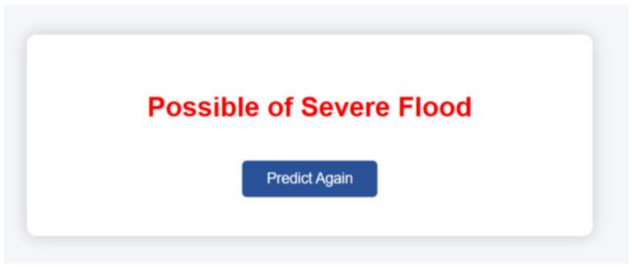
Jun-Sep Rainfall (mm): 43

OUTPUT-1:



Not possible of Severe Flood

OUTPUT-2:



Possible of Severe Flood