Rising Waters: A Machine Learning Approach to Flood Prediction

Team ID: LTVIP2026TMIDS75186

Team Size: 4

Team member: Manthina Subhash Raju

Team Leader: Tatina Deepthi Bala Satya Sri

Team member: Laxman Abhinav Tatipudi

Team member: Malla Veerasiva

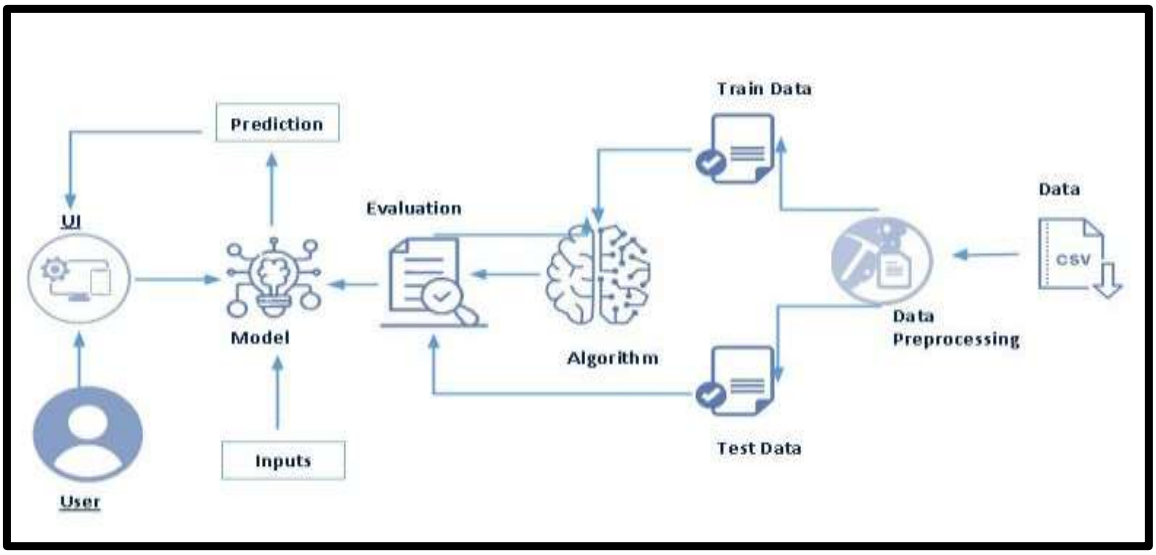
Project Description:

One of the most critical factors affecting a nation's socioeconomic stability and infrastructure is the recurring threat of environmental disasters, particularly flooding. As global climate patterns become increasingly volatile, the process of flood risk evaluation is gaining recognition as a top priority for governments and urban planners worldwide. Just as financial institutions must calculate credit risk to maintain stability, environmental agencies must utilize advanced techniques for risk level calculation to protect communities. Flood risk management is now a primary function of modern disaster-preparedness strategies, serving as a safeguard for both human life and national resources.

The prediction of specific flood events remains a difficult task due to the complex interplay of rainfall intensity, soil saturation, and geographical topography. However, by accurately forecasting rising water levels, authorities can significantly reduce the loss of life and minimize damage to non-mobile assets, such as housing and critical infrastructure. This allows for proactive disaster mitigation and efficient resource allocation, ensuring that emergency responses are both timely and cost-effective. Consequently, the study of machine learning-based flood prediction is essential for creating resilient urban environments.

Machine Learning techniques are exceptionally useful in processing the high-dimensional data required for these environmental predictions. This project employs a suite of classification algorithms—including Decision Tree, Random Forest, KNN, and XGBoost—to analyze historical hydrological datasets. We will rigorously train and test these models to identify the most accurate predictor, which will then be exported in a .pkl format for production use. To ensure the tool is accessible to end-users, we will utilize Flask integration for the web interface.

Technical Architecture:



Pre requisites:

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

- **Anaconda navigator and PyCharm:**
  - Refer the link below to download anaconda navigator
  - Link : https://youtu.be/1ra4zH2G4o0
- **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 scipy” 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 following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Regression and classification
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical data
  - Splitting data into train and test
- Model building
  - Import the model building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating performance of model
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code

Project Structure:

Create the Project folder which contains files as shown below

/your_project_folder

```
|
├── app.py             (Your python code)
├── floods.save        (Model file)
├── transform.save     (Scaler file)
|
└── templates/         <-- Create this folder
    ├── home.html      (The page with the intro)
    ├── intro.html     (The page with the input form)
    ├── noChance.html
```

└── Chance.html

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- rdf.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and training_ibm folder contains IBM deployment files.

Milestone 1: Data Collection

ML depends heavily on data, it is most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used drug200.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

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

Milestone 2: Visualizing and analysing the data

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

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as fivethirtyeight.

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score
```

Activity 2: Read 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.

```
df.describe()
```
0.0s

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
count	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000
mean	29.600000	73.852174	36.286957	2925.487826	27.739130	377.253913	2022.840870	497.636522	218.100870	439.801739	0.139130
std	1.122341	2.947623	4.330158	422.112193	22.361032	151.091850	386.254397	129.860643	62.547597	210.438813	0.347597
min	28.000000	70.000000	30.000000	2068.800000	0.300000	89.900000	1104.300000	166.600000	65.600000	34.200000	0.000000
25%	29.000000	71.000000	32.500000	2627.900000	10.250000	276.750000	1768.850000	407.450000	179.666667	295.000000	0.000000
50%	30.000000	74.000000	36.000000	2937.500000	20.500000	342.000000	1948.700000	501.500000	211.033333	430.600000	0.000000
75%	31.000000	76.000000	40.000000	3164.100000	41.600000	442.300000	2242.900000	584.550000	263.833333	577.650000	0.000000
max	31.000000	79.000000	44.000000	4257.800000	98.100000	915.200000	3451.300000	823.300000	366.066667	982.700000	1.000000

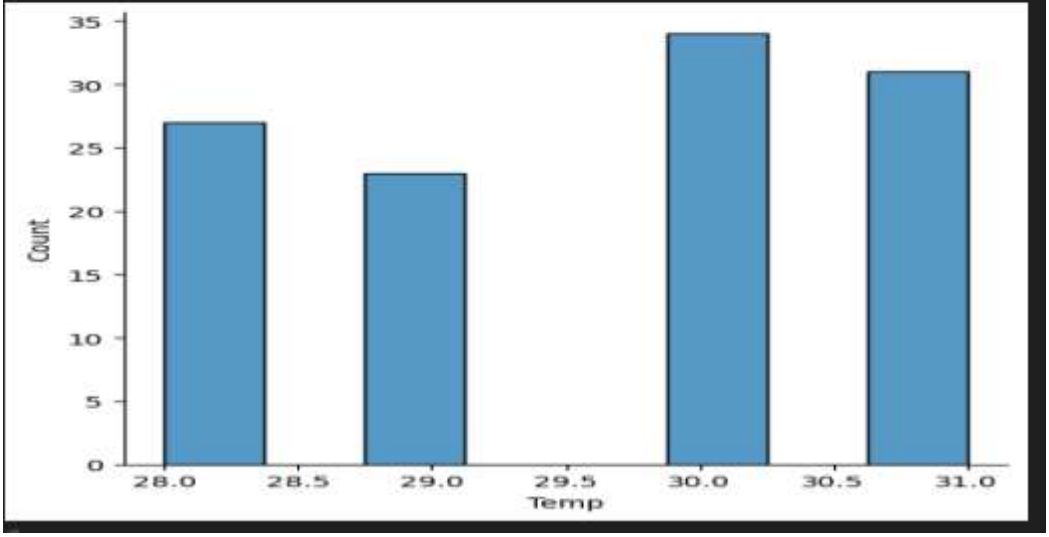
Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

- Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
print(sns.displot(df['Temp']))
```
0.3s

```
<seaborn.axisgrid.FacetGrid object at 0x000002394C3F0500>
```



- In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below graph.
- From the plot we came to know, Applicants income is skewed towards left side, where as credit history is categorical with 1.0 and 0.0

Countplot: -

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot(), so you can compare counts across nested variables.

From the graph we can infer that, gender and education is a categorical variable with 2 categories, from gender column we can infer that 0-category is having more weightage than category-1, while education with 0, it means no education is a underclass when compared with category -1, which means educated.

Activity 4: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used swarm plot from seaborn package.

From the above graph we are plotting the relationship between the Gender, applicants income and loan status of the person

Activity 6: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe()
```
✓ 0.0s

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
count	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000
mean	29.600000	73.852174	36.286957	2925.487826	27.739130	377.253913	2022.840870	497.636522	218.100870	439.801739	0.139130
std	1.122341	2.947623	4.330158	422.112193	22.361032	151.091850	386.254397	129.860643	62.547597	210.438813	0.347597
min	28.000000	70.000000	30.000000	2068.800000	0.300000	89.900000	1104.300000	166.600000	65.600000	34.200000	0.000000
25%	29.000000	71.000000	32.500000	2627.900000	10.250000	276.750000	1768.850000	407.450000	179.666667	295.000000	0.000000
50%	30.000000	74.000000	36.000000	2937.500000	20.500000	342.000000	1948.700000	501.500000	211.033333	430.600000	0.000000
75%	31.000000	76.000000	40.000000	3164.100000	41.600000	442.300000	2242.900000	584.550000	263.833333	577.650000	0.000000
max	31.000000	79.000000	44.000000	4257.800000	98.100000	915.200000	3451.300000	823.300000	366.066667	982.700000	1.000000

Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of 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
- Scaling Techniques
- Splitting dataset into training and test set

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.

Activity 1: Checking for null values

- Let’s find the shape of our dataset first, To find the shape of our data, df.shape method is used. To find the data type, df.info() function is used.

```
print(df.info())
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115 entries, 0 to 114
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Temp         115 non-null    int64
 1   Humidity     115 non-null    int64
 2   Cloud Cover  115 non-null    int64
 3   ANNUAL       115 non-null    float64
 4   Jan-Feb      115 non-null    float64
 5   Mar-May      115 non-null    float64
 6   Jun-Sep      115 non-null    float64
 7   Oct-Dec      115 non-null    float64
 8   avgjune      115 non-null    float64
 9   sub          115 non-null    float64
 10  flood        115 non-null    int64
dtypes: float64(7), int64(4)
memory usage: 10.0 KB
None
```

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling of missing values step.

```
df.isnull().any()
```
✓ 0.0s

```
Temp          False
Humidity      False
Cloud Cover   False
ANNUAL        False
Jan-Feb       False
Mar-May       False
Jun-Sep       False
Oct-Dec       False
avgjune       False
sub           False
flood         False
dtype: bool
```

Activity 2: Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.

```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset becau
# there different type of values in the columns
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)

x_bal = pd.DataFrame(x_bal,columns=names)
```

We will perform scaling only on the input values

Once the dataset is scaled, it will be converted into array and we need to convert it back to dataframe.

Activity : Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
#splitting the dataset in train and test on balnmced datasew
X_train, X_test, y_train, y_test = train_test_split(
     x_bal, y_bal, test_size=0.33, random_state=42)
```

Milestone 4: 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.

Activity 1: Decision tree model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialized and training data is passed

to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
def decisionTree(x_train, x_test, y_train, y_test)
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train)
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Activity 2: Random forest model

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

```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Activity 3: KNN model

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

```
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Activity 4: Xgboost model

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

```
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Now let's see the performance of all the models and save the best model

Activity 5: Compare the model

For comparing the above four models compareModel function is defined.

```
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))
```

✓ 0.0s

```
1.0
1.0
0.9130434782608695
1.0
```

After calling the function, the results of models are displayed as output. From the four model Xgboost is performing well. From the below image, We can see the accuracy of the model. Xgboost is giving the accuracy of 94.7% with training data , 81.1% accuracy for the testing data.so we considering xgboost and deploying this model.

Activity 6: Evaluating performance of the model and saving the model

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to 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

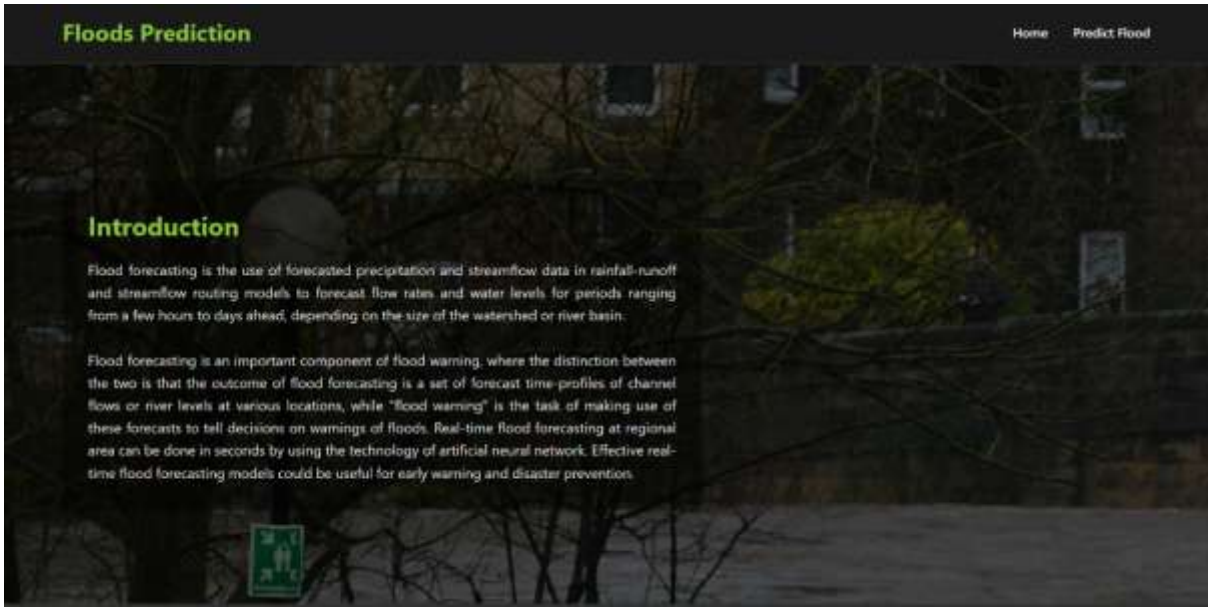
Activity1: Building Html Pages:

For this project create three HTML files namely

- home.html
- predict.html
- submit.html

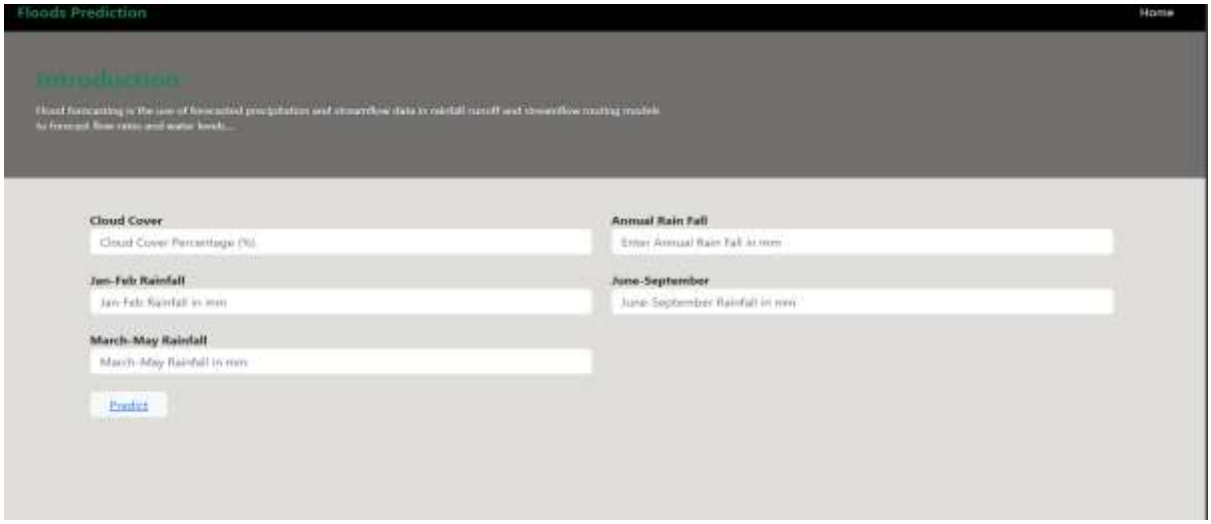
and save them in templates folder.

Let's see how our home.html page looks like:



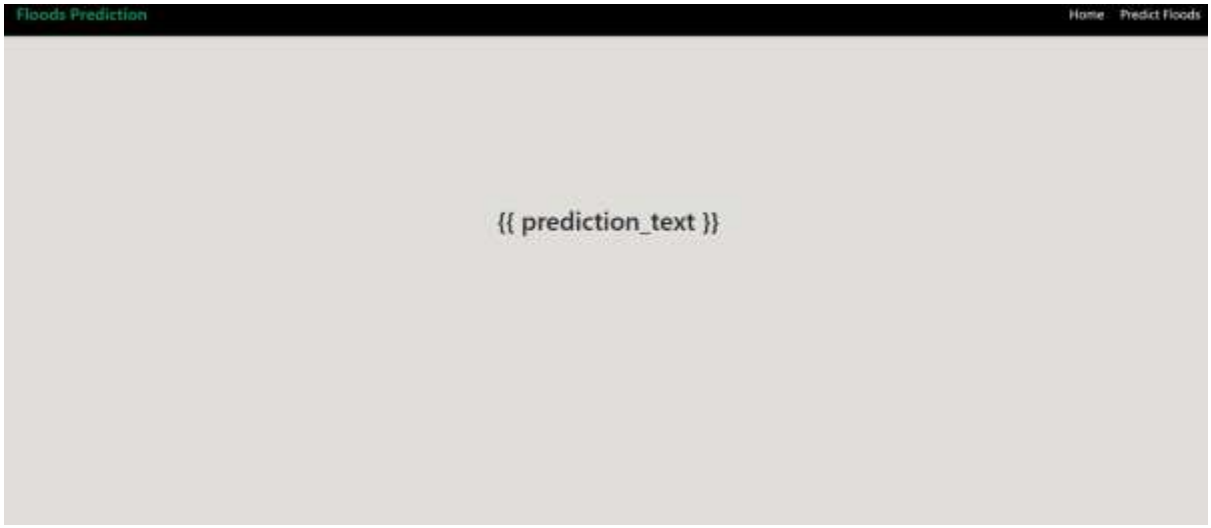
Now when you click on predict button from top right corner you will get redirected to intro.html

Let's look how our predict.html file looks like:



Now when you click on submit button from left bottom corner you will get redirected to predict.html

Lets look how our submit.html file looks like:



Activity 2: Build Python code:

Import the libraries

```
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


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

Render HTML page:

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

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

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@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 severve flood')
    else:
        return render_template('Chance.html',prediction='Possibility of severve flood')
```

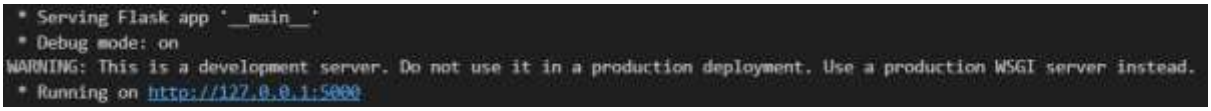
Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

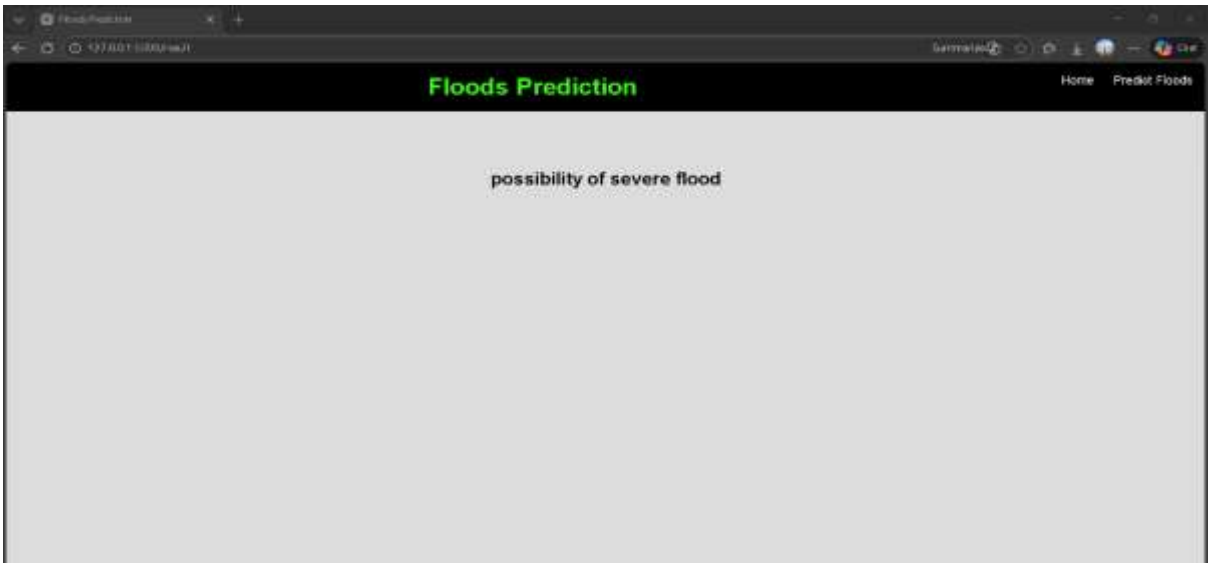
Main Function:

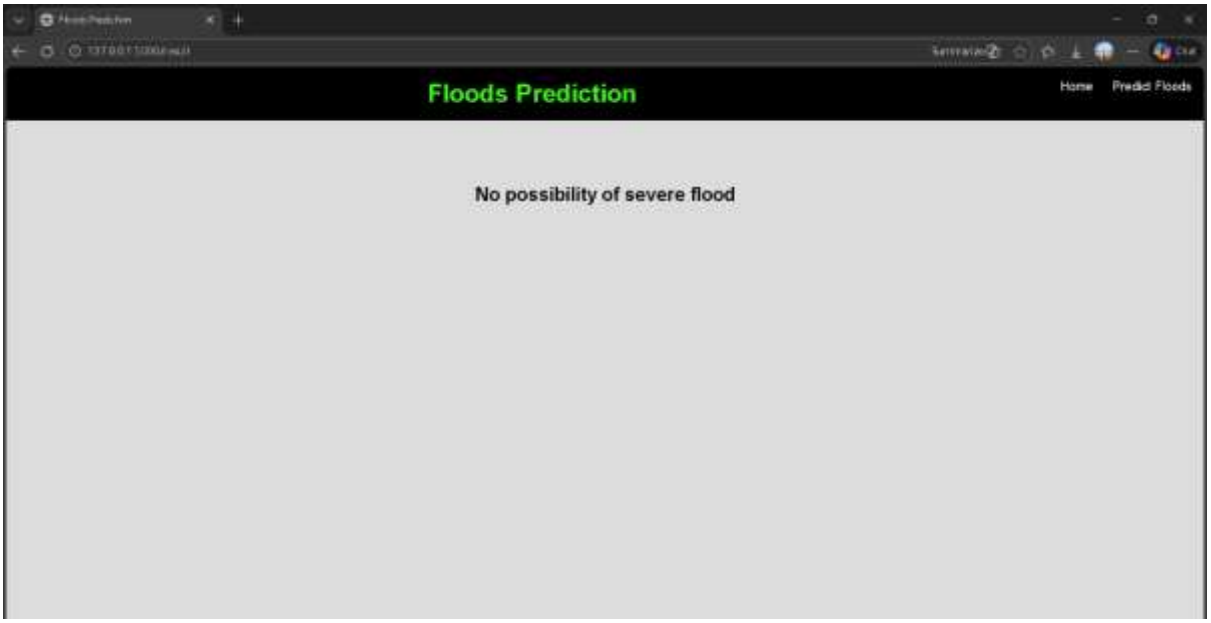
```
if __name__=='__main__':
    app.run(debug=True,use_reloader=False)
```

Activity 3: Run the application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.







Advantages

1. Early Risk Identification

- Predicts flood risk before water level becomes critical.
- Helps authorities prepare evacuation and resource allocation in advance.

2. Data-Driven Decision Making

- Uses historical rainfall and river data instead of guesswork.
- Reduces dependency on purely manual monitoring systems.

3. Scalable Architecture

- Can extend from one district to multiple states.
- Cloud deployment allows handling large environmental datasets.

4. Reduced Human Error

- Automated prediction minimizes subjective judgment.
- Standardized model ensures consistent risk classification.

Limitations

1. Data Dependency

- Poor-quality or missing rainfall data reduces accuracy.
- Inaccurate river gauge readings directly affect predictions.

2. Model Generalization Issues

- Model trained on one region may not perform well in another.
- Sudden climate anomalies (extreme rare events) may not be predicted accurately.

3. Imbalanced Data Problem

- Flood events are rare compared to non-flood days.
- Model may bias toward “No Flood” predictions if not handled properly.

4. Infrastructure Dependency

- Requires stable API connections and database availability.
- Alert system failure may delay warning delivery.

Applications

1. Disaster Management Authorities

- District-level flood monitoring dashboards.
- Real-time risk alerts for evacuation planning.

2. Agricultural Planning

- Farmers can take preventive action before heavy flooding.
- Crop insurance companies can assess risk levels.

3. Urban Planning & Smart Cities

- Flood-prone area mapping for infrastructure planning.
- Drainage system improvement based on risk trends.

4. Government Policy Support

- Long-term flood trend analysis.
- Budget allocation for high-risk regions.

Future Scope

1. Deep Learning Integration

- Use LSTM or GRU for advanced time-series forecasting.
- Improve multi-day ahead prediction accuracy.

2. IoT Sensor Integration

- Real-time river level sensors.
- Automated rainfall station integration.

3. Satellite Image Analysis

- Use remote sensing data to detect water spread.
- Combine CNN models with rainfall prediction.

4. Multi-Hazard Extension

- Extend system to predict landslides.
- Integrate cyclone and storm surge forecasting.

5. Mobile Application

- Direct citizen alert app.
- Location-based flood warnings.

Conclusion

The Rising Waters system demonstrates how machine learning can enhance traditional flood forecasting by analyzing historical and real-time environmental data.

The system improves early warning capability, supports data-driven disaster management, and reduces potential loss of life and property. However, its effectiveness depends heavily on data quality, model tuning, and reliable infrastructure.

With integration of real-time sensors, deep learning models, and scalable cloud deployment, the system can evolve into a comprehensive disaster prediction platform.

GitHub Link: - https://github.com/Subhash-05/Rising-waters-A-Machine-Learning-Approach-to-Flood-Prediction.git

Demo Video Link:

https://drive.google.com/file/d/1pf8rDl8QgeFs7zaK8q3jiBPgY_EK6h6E/view?usp=drivesdk