Rising Waters: A Machine Learning Approach to Flood Prediction

MENTOR NAME :- **SRI L. LANKA LAKSHMI NARAYANA**

TEAM ID :- **LTVIP2026TMIDS46451**

TEAM LEADER

NAME : M.MANIKANTA
MAIL : maniroymullagudi@gmail.com
ID NO : SBAP0040420

TEAM MEMBERS

MEMBER - 1

NAME : MD.AYUB
MAIL : Mdayub37407@gmail.com
ID NO : SBAP0040411

MEMBER - 2

NAME : B.PRUDHVI
MAIL : prudhvibattina272@gmail.com
ID NO : SBAP0040414

Introduction

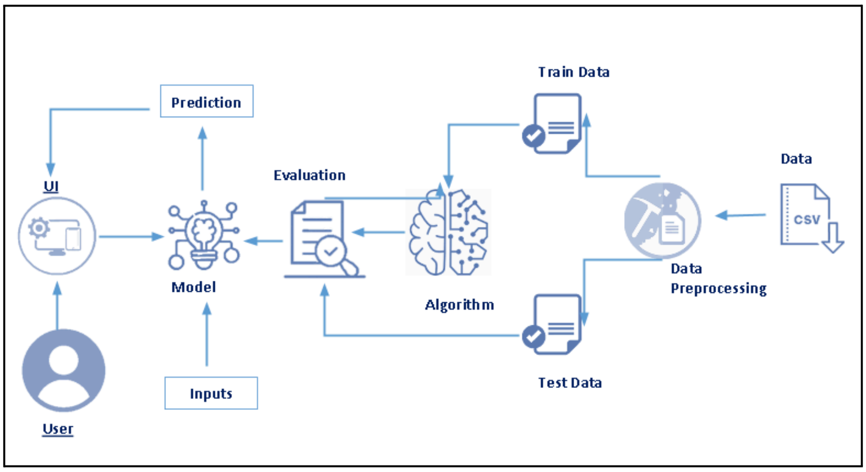
Flood Prediction using Machine Learning is a vital application that aims to forecast and predict flood occurrences with high accuracy. By analyzing historical weather data, river levels, terrain information, and other relevant factors using machine learning algorithms, this project helps in early warning 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



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

Name	Type	Date Modified
Dataset	File Folder	17-08-2021 12:06
flood dataset.xlsx	xlsx File	17-08-2021 12:06
Flask	File Folder	17-08-2021 12:06
templates	File Folder	17-08-2021 12:06
app.py	py File	17-08-2021 12:06
floods.save	save File	17-08-2021 12:06
transform.save	save File	17-08-2021 12:06
IBM scoring end point	File Folder	21-02-2022 17:02
templates	File Folder	21-02-2022 17:02
app.py	py File	17-08-2021 12:06
Training	File Folder	17-08-2021 12:06
Floods.ipynb	ipynb File	17-08-2021 12:06
Floods prediction using machine learning.docx	docx File	21-02-2022 11:44

- **We are building a Flask Application that needs HTML pages "home.html", "index.html" stored in the templates folder and a python script app.py for server-side scripting**
- **The model is built in the notebook floods.ipynb**
- **We need the model which is saved and the saved model in this content is floods.save and transform.save**
- **The templates mainly used here are "home.html", "chance.html", "no chance.html", "index.html" for showcasing the UI**
- **The flask app is denoted as app.py**
- IBM scoring end point consist templates and 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.

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 required libraries
import numpy as np #for dealing high demensional data
import pandas as pd #to do statistical data analysis
import matplotlib.pyplot as plt #for 2D visualization
import seaborn as sns #High end data visualization
```

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.**

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

Uni-variate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distribution plot, box plot.

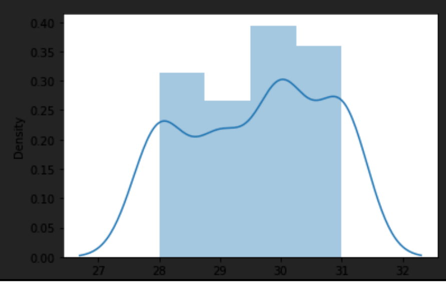
Distribution plot :

- **The distribution plot is suitable for comparing range and distribution for groups of numerical data. Data are plotted as value points along an axis.**
- **it is used to identify, what kind of distribution does the data follow**

```
In [5]: print(sns.distplot(dataset['Temp']))

C:\Users\SmartbridgePC\anaconda4\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `
histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

AxesSubplot(0.125,0.125;0.775x0.755)
```



- **From the above graph, we can infer that the column temp follows some kind of normal distribution, it means the data is normal is almost normal**

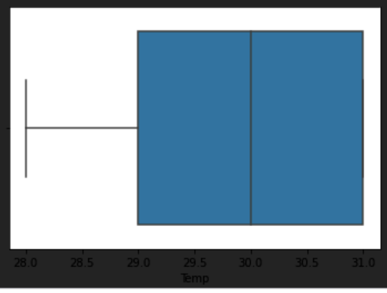
Boxplot:

- **Box plot is used on the length of service and average training score feature. Length of services feature has more outliers. The model should not be built without handling the outliers. Here, outliers are handled by the capping method. Capping will be discussed on data pre-processing.**

```
In [6]: print(sns.boxplot(dataset['Temp']))
```

```
AxesSubplot(0.125,0.125;0.775x0.755)

C:\Users\SmartbridgePC\anaconda4\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



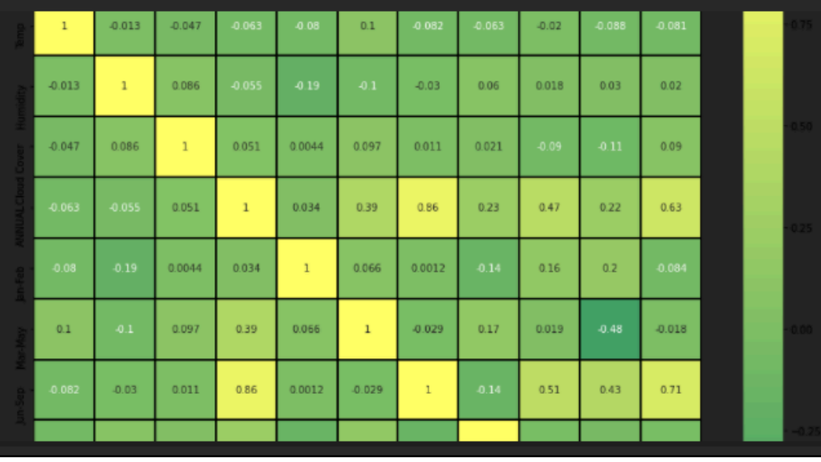
Multivariate Analysis

In simple words, multivariate analysis is to find the relation between multiple features.

Heat map :

- **A heat map is a data visualization technique that shows magnitude of a phenomenon as colour in two dimensions. The variation in colour may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.**

```
In [72]: import seaborn as sns
fig=plt.gcf()
fig.set_size_inches(15,15)
fig=sns.heatmap(dataset.corr(),annot=True,cmap='summer',
                linewidths=1,linecolor='k',square=True,
                mask=False, vmin=-1, vmax=1,
                cbar_kws={"orientation": "vertical"},cbar=True)
```



- **From the graph, we can identify less correlated values and can drop those variables.**

Descriptive Analysis

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

```
#check the first 5 observations
dataset.head()
```

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
0	29	70	30	3248.6	73.4	386.2	2122.8	666.1	274.866667	649.9	0
1	28	75	40	3326.6	9.3	275.7	2403.4	638.2	130.300000	256.4	1
2	28	75	42	3271.2	21.7	336.3	2343.0	570.1	186.200000	308.9	0
3	29	71	44	3129.7	26.7	339.4	2398.2	365.3	366.066667	862.5	0
4	31	74	40	2741.6	23.4	378.5	1881.5	458.1	283.400000	586.9	0

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(dataset.info())
```

```
<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
```

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.

```
dataset.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Temp	115.0	29.600000	1.122341	28.0	29.000000	30.000000	31.000000	31.000000
Humidity	115.0	73.852174	2.947623	70.0	71.000000	74.000000	76.000000	79.000000
Cloud Cover	115.0	36.286957	4.330158	30.0	32.500000	36.000000	40.000000	44.000000
ANNUAL	115.0	2925.487826	422.112193	2068.8	2627.900000	2937.500000	3164.100000	4257.800000
Jan-Feb	115.0	27.739130	22.361032	0.3	10.250000	20.500000	41.600000	98.100000
Mar-May	115.0	377.253913	151.091850	89.9	276.750000	342.000000	442.300000	915.200000
Jun-Sep	115.0	2022.840870	386.254397	1104.3	1768.850000	1948.700000	2242.900000	3451.300000
Oct-Dec	115.0	497.636522	129.860643	166.6	407.450000	501.500000	584.550000	823.300000
avgjune	115.0	218.100870	62.547597	65.6	179.666667	211.033333	263.833333	366.066667
sub	115.0	439.801739	210.438813	34.2	295.000000	430.600000	577.650000	982.700000
flood	115.0	0.139130	0.347597	0.0	0.000000	0.000000	0.000000	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.

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

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
```

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

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

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

- Capping: Replacing the outliers with upper bound values.

Note: In our Dataset all the values are in the same range, so outliers replacing is not necessary.

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.

Note: In our dataset, there is no categorical data type, so we can skip this step.

Splitting the Dataset into Dependent and Independent variables.

In machine learning, the concept of the dependent variable (y) and independent variables(x) is important to understand. Here, the Dependent variable is nothing but output in the dataset and the independent variable is all inputs in the dataset. We can denote with any symbol (alphabets). In our dataset, we can say that class is the dependent variable and all other columns are independent. But in order to select the independent columns, we will be selecting only those columns which are highly correlated and some value to our dependent column.

- **With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.**

- **Let's create out independent and dependent variables:**

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

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

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

Split the dataset into Train set and Test set

Now split our dataset into a train set and test using train_test_split class from sci-kit learn library.

- **Train_test_split: used for splitting data arrays into training data and for testing data.**

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

Feature scaling

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

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

- Save the StandardScaler

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

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

```
from sklearn import tree
from sklearn import ensemble
from sklearn import neighbors
import xgboost
```

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

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

Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test

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

```
from sklearn import tree
from sklearn import ensemble
from sklearn import neighbors
import xgboost
```

```
dtree = tree.DecisionTreeClassifier()
Rf = ensemble.RandomForestClassifier()
knn = neighbors.KNeighborsClassifier()
xgb = xgboost.XGBClassifier()
```

```
dtree.fit(x_train,y_train)
Rf.fit(x_train,y_train)
knn.fit(x_train,y_train)
xgb.fit(x_train,y_train)
```

KNN model

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


```
from sklearn import tree
from sklearn import ensemble
from sklearn import neighbors
import xgboost
```

```
dtree = tree.DecisionTreeClassifier()
Rf = ensemble.RandomForestClassifier()
knn = neighbors.KNeighborsClassifier()
xgb = xgboost.XGBClassifier()
```

```
dtree.fit(x_train,y_train)
Rf.fit(x_train,y_train)
knn.fit(x_train,y_train)
xgb.fit(x_train,y_train)
```

Xgboost model

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

Fit the model using x_train, y_train data

```
from sklearn import tree
from sklearn import ensemble
from sklearn import neighbors
import xgboost
```

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

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

Compare the model

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

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

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

```
0.9655172413793104
0.9655172413793104
0.896551724137931
0.9655172413793104
```

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

To get deep knowledge of confusion matrix and classification reports refer to the link

Evaluating performance of the model

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

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

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

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

0.9655172413793104

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

0.75

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

1.0
```

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.

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

```
['floods.save']
```

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

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 "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.

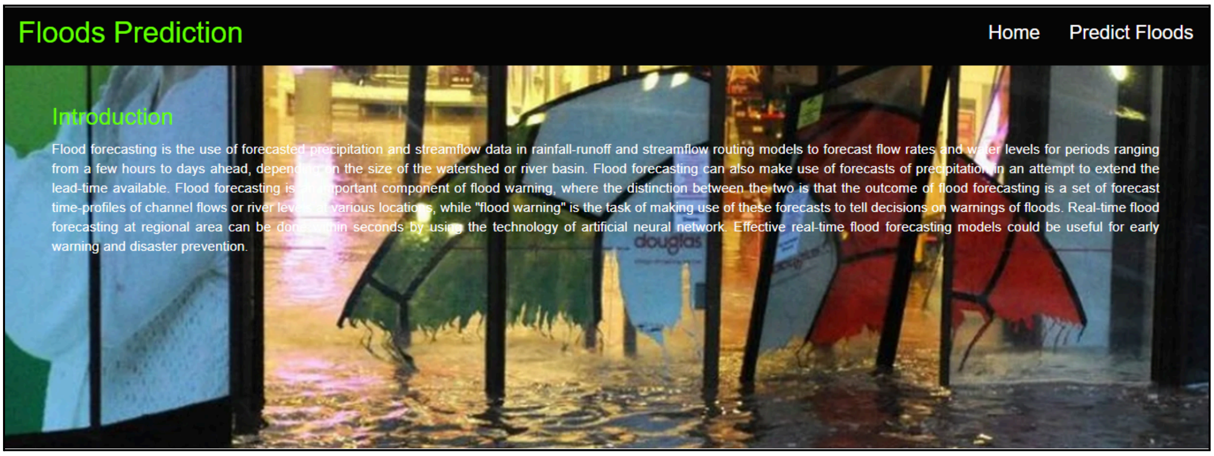
Here, we created 4 html pages- home.html, image.html, imageprediction.html, intro.html.

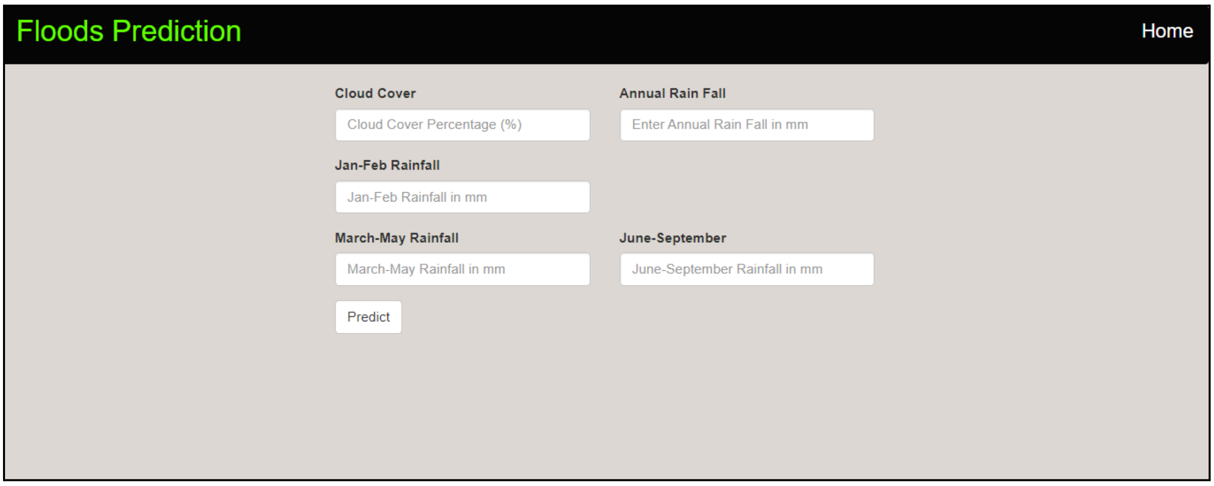
For more information regarding HTML see the link

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

- **Link: link**

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





Build python code

- **Let us build flask file 'app.py' which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.**
- **App starts running when "__name__" constructor is called in main.**
- **render_template is used to return HTML file.**
- **"GET" method is used to take input from the user.**
- **"POST" method is used to display the output to the user.**

- We will be using python for server side scripting. Let's see step by step process for writing backend code.

Importing Libraries

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

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

Create Flask app and Load our model file

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

Routing to the HTML Page

- Here we will be using the declared constructor to route to the HTML page that we have created earlier.
- In the above example, the '/' URL is bound with the home.html function. Hence, when the home page of the webserver is opened in the browser, the HTML page is rendered. Whenever you click on the Intro button from the home page, the intro.html page will be rendered.

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

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

Route the prediction on UI

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

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

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

Main Function

- This is used to run the application in localhost

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

Run the application

- Open anaconda prompt from start menu.
- Navigate to the folder where your app.py resides.
- Now type the “python app.py” command.
- 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.

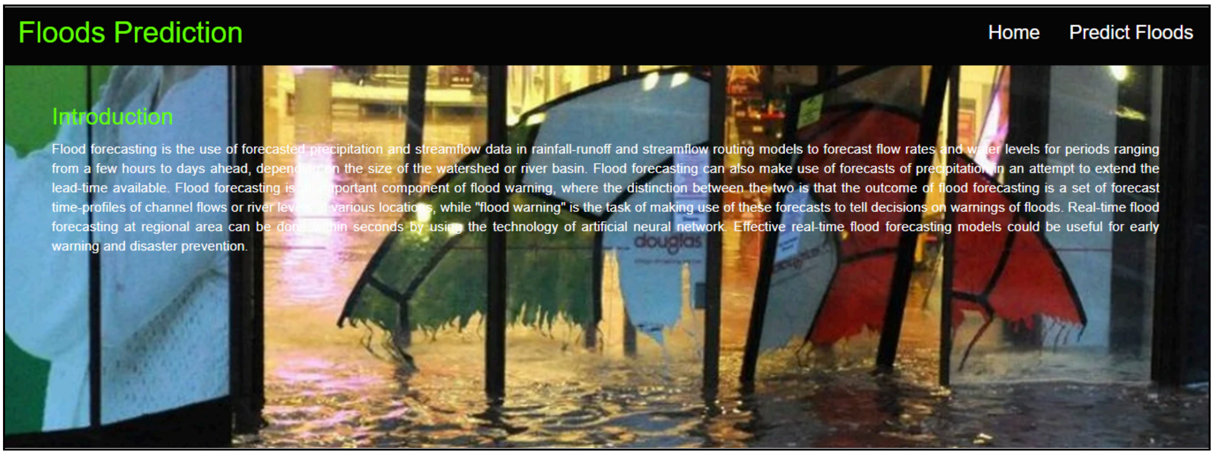
```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 135-972-108
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:5000/) where you can view your web page.

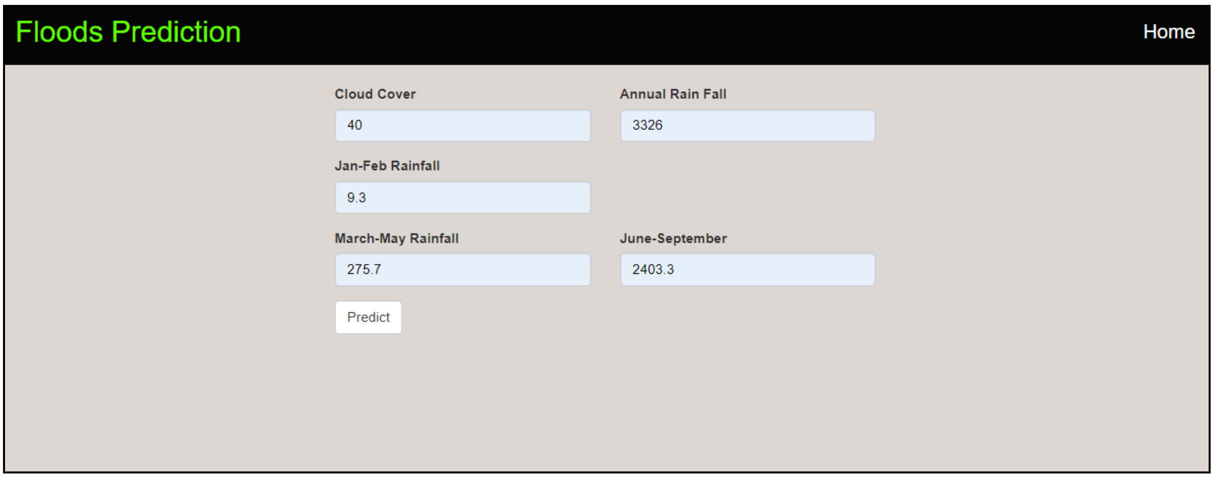
Click on Predict Floods and submit the values to get the floods prediction result on the web page.

Output:

- **This is the home page of floods prediction.**

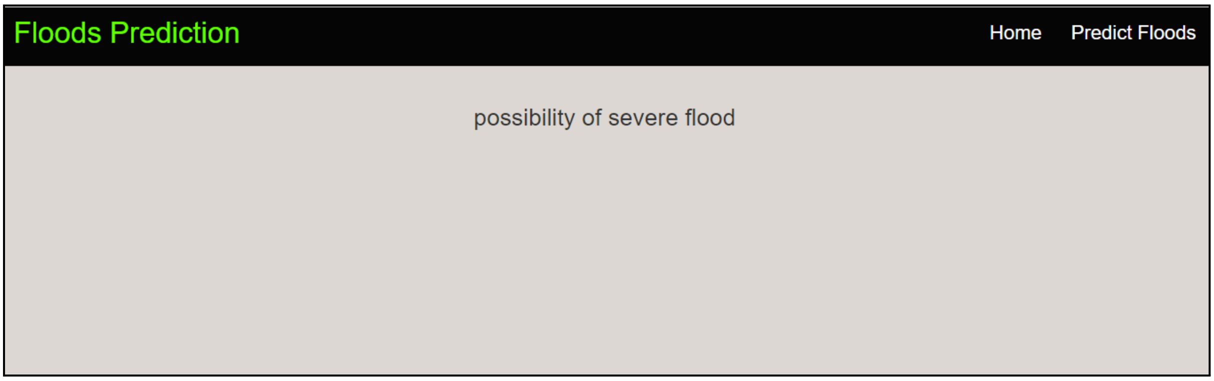
- 

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

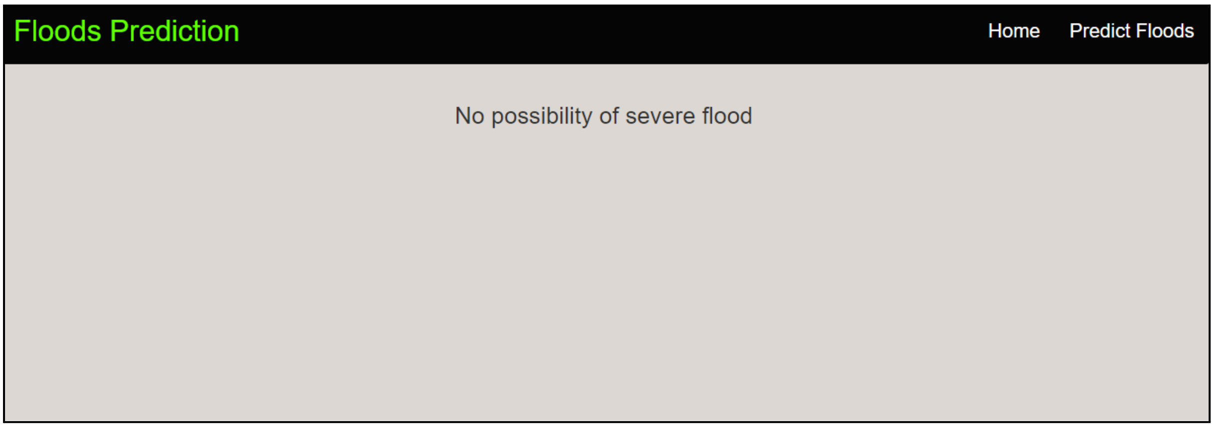


- **On the prediction page, users will get the output based on the inputs they give on the prediction page.**

Output1:


Floods Prediction
Home
Predict Floods
possibility of severe flood

Output 2:


Floods Prediction
Home
Predict Floods
No possibility of severe flood

COMMENTS

FIRST AND FOREMOST, I SINCERELY GRATITUDE TO OUR ESTEEMED INSTITUTE SRI VASAVI DEGREE COLLEGE, FOR GIVING ME THIS OPPORTUNITY TO FULFILL OUR WARM DREAM TO BECOME A GRADUATE. OUR SINCERE GRADITUDE TO OUR LONG-TERM INTERNSHIP GUIDE **SRI L LAKSHMI NARAYANA**, LECTURER DEPARTMENT OF COMPUTER SCIENCE FOR TIMELY COOPERATION AND VALUABLE SUGGESTIONS WHILE CARRYING OUT THIS INTERNSHIP.

I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO **SRI L LAKSHMI NARAYANA**, HOD IN COMPUTER SCIENCE FOR PERMITTING ME TO DO MY PROJECT INTERNSHIP. I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO **SRI M RAMA KRISHNA**, PRINCIPAL FOR PROVIDING A FAVOURABLE ENVIRONMENT AND SUPPORTING ME DURING THE DEVELOPMENT OF THIS INTERNSHIP.

THANK YOU,SMART BRIDGE

----MULLAPUDI MANIKANTA
TEAM LEADER

THE END

SIGNATURE OF THE HOD

SIGNATURE OF THE PRINCIPAL