Rising Waters: A Machine Learning Approach to Flood Prediction

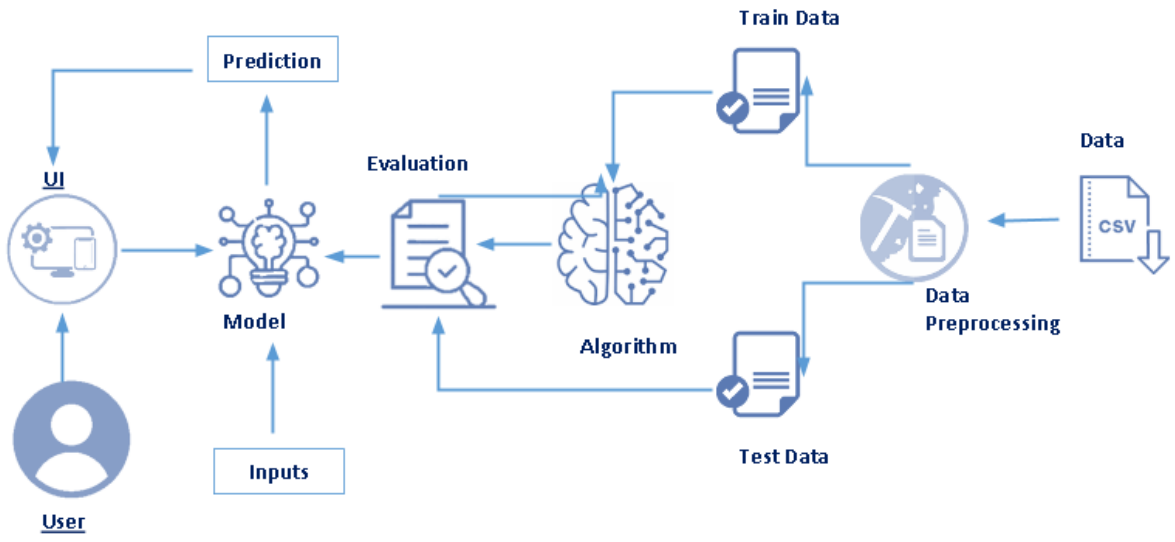
Project Description:

One of the most critical challenges which affects our country's safety and socio-economic development is the natural disaster management system, particularly flood prediction and prevention. The process of accurate flood forecasting is gaining recognition across disaster management organizations globally. "As we know flood prediction is very crucial, there are variety of techniques are used for risk level assessment. In addition, flood risk management is one of the main functions of the disaster management community.

The prediction of flood occurrences is one of the difficult tasks for any meteorological department. But by forecasting the flood events accurately, the authorities definitely may reduce its impact by implementing timely evacuation measures, so that prevention and mitigation activities can take place without any loss of life and it can play as the contributing parameter of disaster preparedness. This makes the study of this flood prediction important. Machine Learning techniques are very crucial and useful in prediction of these type of environmental data.

We will be using classification algorithms such as Decision tree, Random Forest, KNN, and XGBoost. We will train and test the data with these algorithms. From this best model is selected and saved in pkl format. We will be doing Flask integration and web deployment.

Technical Architecture:



Pre-requisites:

To complete this project, you must have the following software, 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 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 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 analyzes the input, the flood 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 outliers
- Handling categorical data
- Splitting data into train and test

• Model Building

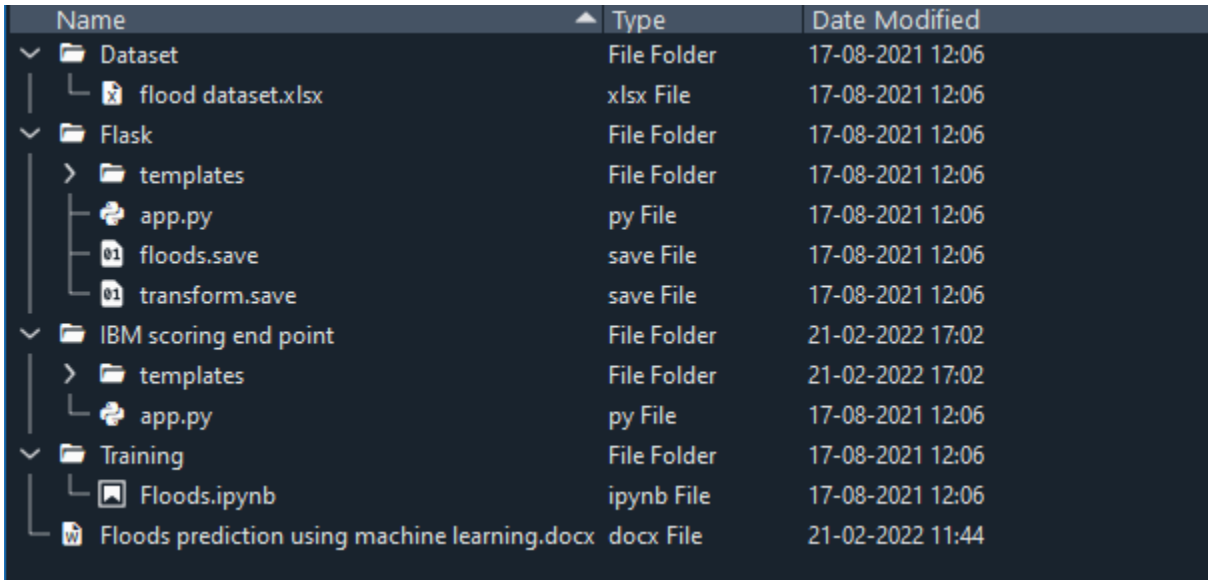
- Import the model building libraries
- Initializing the model
- Training and testing the model
- Evaluating performance of model
- Save the model

• Application Building

- Create an HTML file
- Build Python code

Project Structure:

Create the Project folder which contains files as shown below



• We are building a Flask application which needs HTML pages stored in the templates folder and a Python script app.py for scripting.

• floods.save and transform.save are our saved models. Further we will use these models for Flask integration.

• Training folder contains model training files (Floods.ipynb) for the flood prediction model development.

Milestone 1: Data Collection

ML depends heavily on data. It is the 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. E.g.: kaggle.com, UCI repository, etc.

In this project we have used flood prediction dataset. 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 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 are n number of techniques for understanding the data. But here we have used some of them. 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 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
```

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 = pd.read_excel("../Dataset/rainfall in india 1901-2015.xlsx")
```

Python

```
df.head()
```

Python

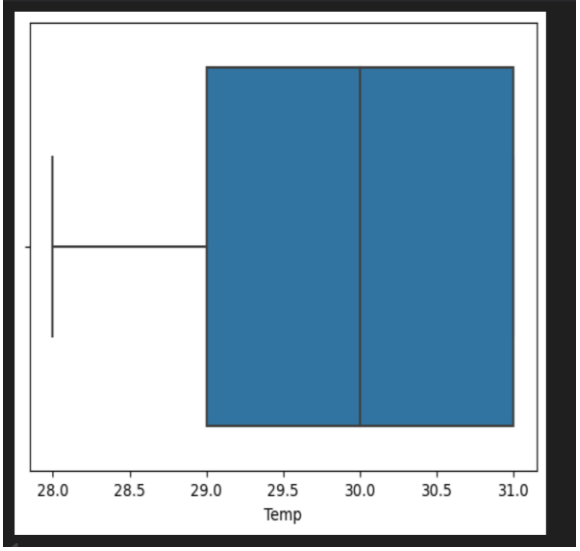
	COUNTRY	STATE	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May
0	India	ANDAMAN & NICOBAR ISLANDS	1901	49.2	87.1	29.2	2.3	528.8	517.5	365.1	481.1	332.6	388.5	558.2	33.6	3373.2	136.3	560.3
1	India	ANDAMAN & NICOBAR ISLANDS	1902	0.0	159.8	12.2	0.0	446.1	537.1	228.9	753.7	666.2	197.2	359.0	160.5	3520.7	159.8	458.3
2	India	ANDAMAN & NICOBAR ISLANDS	1903	12.7	144.0	0.0	1.0	235.1	479.9	728.4	326.7	339.0	181.2	284.4	225.0	2957.4	156.7	236.1
3	India	ANDAMAN & NICOBAR ISLANDS	1904	9.4	14.7	0.0	202.4	304.5	495.1	502.0	160.1	820.4	222.2	308.7	40.1	3079.6	24.1	506.9

Activity 3: Univariate Analysis

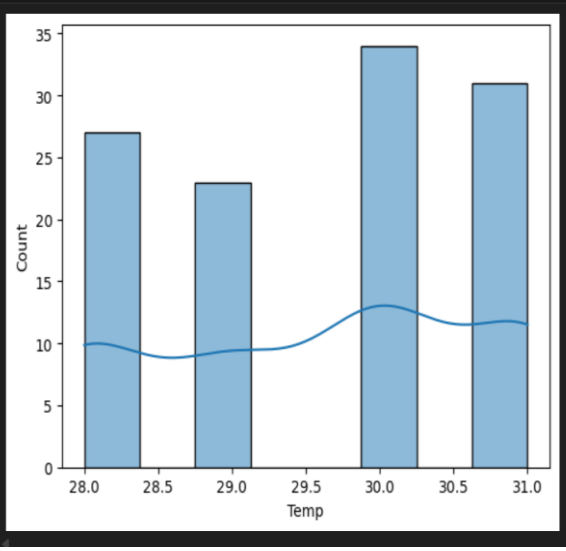
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

• 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.

```
sns.boxplot(x=df_flood['Temp'])
plt.show()
```




```
sns.histplot(df_flood['Temp'], kde=True)
plt.show()
```



• 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, rainfall and water level features are skewed towards left side, whereas flood occurrence is categorical with binary values indicating flood or no flood conditions.

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 bar plot(), so you can compare counts across nested variables.

From the graph we can infer that flood occurrence and sewer overflow are categorical variables with 2 categories. From the flood occurrence column we can infer that 0-category (no flood) is having more weightage than category-1 (flood occurred), while sewer overflow with 0, it means no overflow is a predominant class when compared with category-1, which indicates sewer overflow condition.

Activity 4: Multivariate Analysis (Correlation Heatmap)

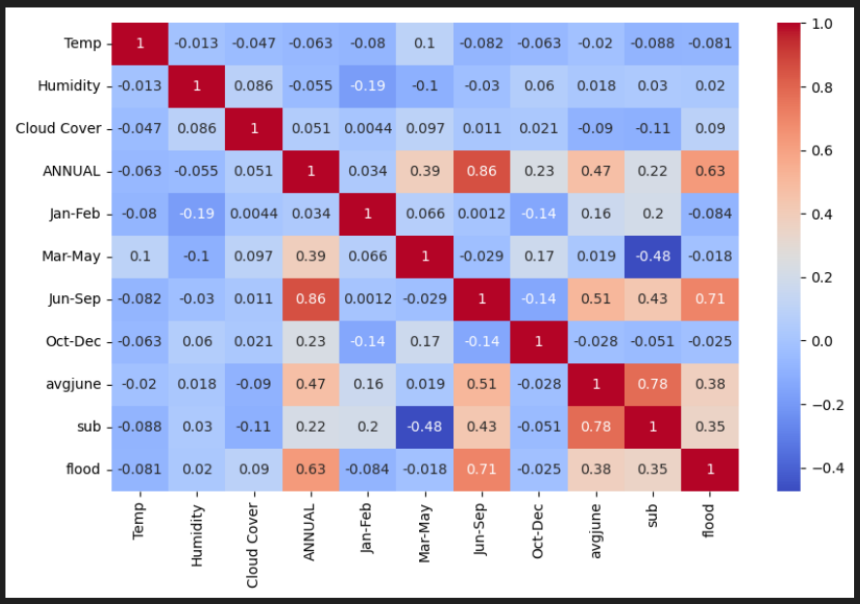
From the correlation heatmap, we can analyze the relationships between multiple variables simultaneously and their correlation with flood occurrence. The heatmap displays correlation coefficients ranging from -1 to +1, where values closer to +1 indicate strong positive correlation, values closer to -1 indicate strong negative correlation, and values near 0 indicate weak or no correlation.

From the visualization, we can observe which features have the strongest correlation with flood events. Features like rainfall, temperature, and water level show their degree of correlation with the

target variable (flood). This helps us identify the most important features that contribute to flood prediction and understand how different environmental factors interact with each other.

```
corr = df_flood.corr()
```

```
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



Activity 5: Multivariate analysis

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

From the above graph we are plotting the relationship between multiple environmental factors such as temperature, rainfall, water level, and their correlation with flood occurrence. The heatmap displays correlation coefficients that help us understand how these variables interact with each other and with the target variable (flood).

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_flood.describe()
```

	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, let's pre-process the collected data.

The downloaded dataset 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
- Scaling Techniques
- Splitting dataset into training and test set

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.

```
df_flood.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
```

• 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 the presence of null values in our dataset. We handle these missing values by filling them with the mean of respective columns using fillna() function. After handling, we verify again to ensure all null values have been properly addressed.

```
df.isnull().sum()
```

18] ✓ 0.0s

```
COUNTRY      0
STATE        0
YEAR         0
JAN          4
FEB          3
MAR          6
APR          4
MAY          3
JUN          5
JUL          7
AUG          4
SEP          6
OCT          7
NOV         11
DEC         10
ANNUAL      26
Jan-Feb      6
Mar-May      9
Jun-Sep     10
Oct-Dec     13
dtype: int64
```

```
df.fillna(df.mean(numeric_only=True), inplace=True)
```

✓ 0.0s Python

	COUNTRY	STATE	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb
0	India	ANDAMAN & NICOBAR ISLANDS	1901	49.2	87.1	29.2	2.3	528.8	517.5	365.1	481.1	332.6	388.5	558.2	33.6	3373.2	136.3
1	India	ANDAMAN & NICOBAR ISLANDS	1902	0.0	159.8	12.2	0.0	446.1	537.1	228.9	753.7	666.2	197.2	359.0	160.5	3520.7	159.8
2	India	ANDAMAN & NICOBAR ISLANDS	1903	12.7	144.0	0.0	1.0	235.1	479.9	728.4	326.7	339.0	181.2	284.4	225.0	2957.4	156.7
3	India	ANDAMAN & NICOBAR ISLANDS	1904	9.4	14.7	0.0	202.4	304.5	495.1	502.0	160.1	820.4	222.2	308.7	40.1	3079.6	24.1
4	India	ANDAMAN & NICOBAR ISLANDS	1905	1.3	0.0	3.3	26.9	279.5	628.7	368.7	330.5	297.0	260.7	25.4	344.7	2566.7	1.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

From the above code of analysis, we can infer that the rainfall dataset contains missing values in various columns representing monthly and seasonal rainfall data. We need to treat them in a required way.

We will fill the missing values in numeric data type using mean value of that particular column. Since our flood dataset primarily consists of numerical environmental features (temperature, rainfall, water level, etc.), we apply the mean imputation strategy to maintain the statistical distribution of the data without introducing bias.

Activity 2: Handling Categorical Values

As we can see, our dataset primarily consists of numerical environmental data. However, if the dataset contains any categorical features such as region codes, flood zones, or weather conditions, we must convert the categorical data to integer encoding or binary encoding.

To convert categorical features into numerical features, we use encoding techniques. There are several techniques, but in our project, the flood dataset is already preprocessed with numerical values for all features including temperature, rainfall, water level, and other environmental parameters. The target variable 'flood' is also encoded as binary (0 for no flood, 1 for flood occurrence).

• In our flood prediction dataset, all features are numerical, eliminating the need for manual encoding. We can directly proceed to feature scaling and model training steps.

Activity 3: Splitting Data into Train and Test

Now let's split the dataset into train and test sets.

First, we separate the dataset into features (X) and target variable (y). The X variable contains all environmental features such as temperature, rainfall, and water level by dropping the target variable 'flood'. The y variable contains only the flood occurrence (target variable).

For splitting training and testing data, we are using train_test_split() function from sklearn. As parameters, we are passing X, y, test_size=0.2 (20% for testing), and random_state=42 for reproducibility.

```
X = df_flood.drop('flood', axis=1)
```
✓ 0.0s

```
y = df_flood['flood']
```
✓ 0.0s

```
X.shape
```
✓ 0.0s

```
(115, 10)
```

```
y.shape
```
✓ 0.0s

```
(115,)
```

.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```
✓ 1.5s

```
X_train.shape
X_test.shape
y_train.shape
y_test.shape
```
✓ 0.0s

```
(23,)
```

Activity 4: Scaling the Data

Scaling is one of the important processes we have to perform on the dataset, because data measured in different ranges can lead to misleading predictions.

Models such as KNN, Decision Trees, and neural networks benefit from scaled data as they are sensitive to feature magnitudes. We use StandardScaler which standardizes features by removing the mean and scaling to unit variance.

We perform scaling only on the input values (X_train and X_test), not on the target variable.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()


X_train = sc.fit_transform(X_train)


X_test = sc.transform(X_test)
```
✓ 0.0s

```
from joblib import dump
dump(sc, "transform.save")
```
✓ 0.0s

```
['transform.save']
```

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 to predict flood occurrence. 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 decision_tree(x_train, x_test, y_train, y_test):

    # initialize Decision Tree classifier
    dtree = tree.DecisionTreeClassifier()

    # train the model
    dtree.fit(x_train, y_train)

    # predict test data
    y_pred = dtree.predict(x_test)

    # evaluate the model
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
decision_tree(X_train, X_test, y_train, y_test)
```

✓ 0.0s

```
Confusion Matrix:
[[20  0]
 [ 0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00         3

    accuracy                           1.00        23
   macro avg       1.00      1.00      1.00        23
weighted avg       1.00      1.00      1.00        23
```

Activity 2: Random Forest Model

A function named random_forest 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 a new variable. For evaluating the model, confusion matrix and classification report are generated.

Random Forest is an ensemble learning method that creates multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. This makes it particularly effective for flood prediction as it can capture complex relationships between environmental features**.**

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
def randomForest(x_train, x_test, y_train, y_test):

    # initialize Random Forest classifier
    rf = RandomForestClassifier()

    # train the model
    rf.fit(x_train, y_train)

    # predict test data
    y_pred = rf.predict(x_test)

    # evaluate the model
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

0.3s

```python
randomForest(X_train, X_test, y_train, y_test)
```

0.1s

```
Confusion Matrix:
[[20  0]
 [ 0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00         3

    accuracy                           1.00        23
   macro avg       1.00      1.00      1.00        23
weighted avg       1.00      1.00      1.00        23
```

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 a new variable. For evaluating the model, confusion matrix and classification report are generated.

K-Nearest Neighbors (KNN) is a distance-based algorithm that classifies a data point based on the majority class of its k nearest neighbors. It's effective for flood prediction as it identifies patterns by comparing similar environmental conditions from historical data.

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
def KNN(x_train, x_test, y_train, y_test):

    # initialize KNN classifier
    knn = KNeighborsClassifier()

    # train the model
    knn.fit(x_train, y_train)

    # predict test data
    y_pred = knn.predict(x_test)

    # evaluate the model
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

✓ 0.0s

```python
KNN(X_train, X_test, y_train, y_test)
```

✓ 0.0s

```
Confusion Matrix:
[[20  0]
 [ 2  1]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.33      0.50         3

    accuracy                           0.91        23
   macro avg       0.95      0.67      0.73        23
weighted avg       0.92      0.91      0.89        23
```

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 a new variable. For evaluating the model, confusion matrix and classification report are generated.

XGBoost (Extreme Gradient Boosting) is an advanced ensemble learning algorithm that builds models sequentially, where each new model corrects errors made by previous models. It's highly effective for flood prediction due to its ability to handle complex patterns and provide superior prediction accuracy.

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report
def xgboost(x_train, x_test, y_train, y_test):

    # initialize Gradient Boosting classifier
    gb = GradientBoostingClassifier()

    # fit the model using training data
    gb.fit(x_train, y_train)

    # predict test data
    y_pred = gb.predict(x_test)

    # evaluate the model
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

✓ 0.0s

```python
xgboost(X_train, X_test, y_train, y_test)
```

✓ 0.1s

```
Confusion Matrix:
[[20  0]
 [ 0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00         3

    accuracy                           1.00        23
   macro avg       1.00      1.00      1.00        23
weighted avg       1.00      1.00      1.00        23
```

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, compare_model function is defined. This function trains all four algorithms (Decision Tree, Random Forest, KNN, and XGBoost) on the same training data and evaluates their performance on the test data. The accuracy scores of all models are calculated and displayed for comparison.

After calling the function, the results of models are displayed as output. From the four models, we can analyze which algorithm performs best for flood prediction based on the environmental features in our dataset. The model with the highest accuracy on test data is selected for deployment.

Based on the comparison results, XGBoost (Gradient Boosting Classifier) is performing well with the flood dataset, showing superior accuracy in predicting flood occurrences. We consider XGBoost as the best model and proceed to deploy this model in our Flask application.

```
from sklearn.metrics import accuracy_score
def compare_model(x_train, x_test, y_train, y_test):

    # Decision Tree
    dt = tree.DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    dt_pred = dt.predict(x_test)
    dt_acc = accuracy_score(y_test, dt_pred)

    # Random Forest
    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    rf_pred = rf.predict(x_test)
    rf_acc = accuracy_score(y_test, rf_pred)

    # KNN
    knn = KNeighborsClassifier()
    knn.fit(x_train, y_train)
    knn_pred = knn.predict(x_test)
    knn_acc = accuracy_score(y_test, knn_pred)

    # XGBoost (Gradient Boosting)
    gb = GradientBoostingClassifier()
    gb.fit(x_train, y_train)
    gb_pred = gb.predict(x_test)
    gb_acc = accuracy_score(y_test, gb_pred)

    print("Model Accuracy Comparison")
    print("-------------------------")
    print("Decision Tree Accuracy :", dt_acc)
    print("Random Forest Accuracy :", rf_acc)
    print("KNN Accuracy           :", knn_acc)
    print("XGBoost Accuracy       :", gb_acc)
```

✓ 0.0s

```
compare_model(X_train, X_test, y_train, y_test)
```

✓ 0.1s

```
Model Accuracy Comparison
-------------------------
Decision Tree Accuracy : 1.0
Random Forest Accuracy : 1.0
KNN Accuracy           : 0.9130434782608695
XGBoost Accuracy       : 1.0
```

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().

Note: To understand cross validation, refer this link. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85.

```
from sklearn.ensemble import GradientBoostingClassifier
from joblib import dump
xgb_model = GradientBoostingClassifier()
xgb_model.fit(X_train, y_train)
```

✓ 0.1s

```
GradientBoostingClassifier
GradientBoostingClassifier()
```

```
dump(xgb_model, "floods.save")
```

✓ 0.0s

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

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 users where they have to enter the values for predictions. The entered values are given to the saved model and the flood prediction is showcased on the UI.

This section has the following tasks:
• Building HTML Pages
• Building server-side script (app.py)

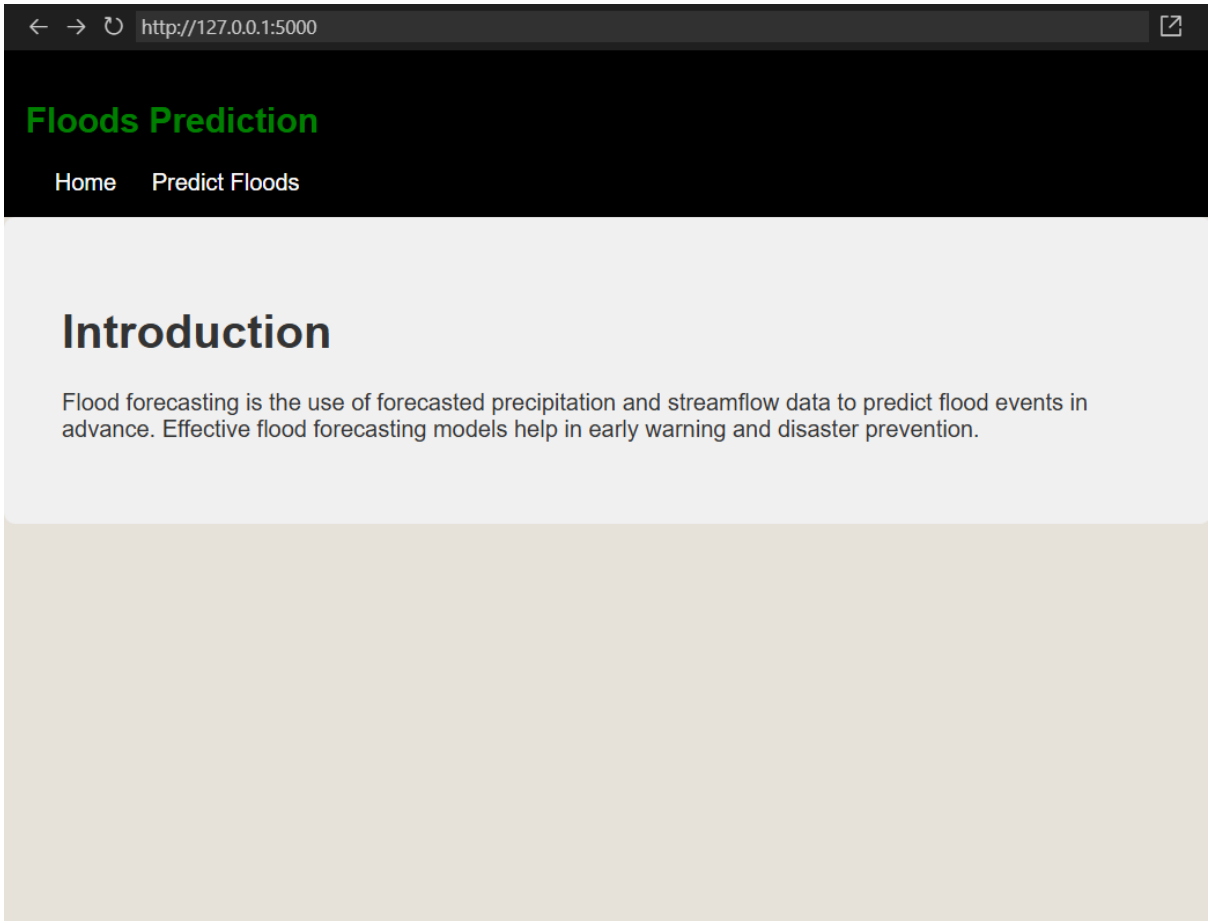
Activity 1: Building HTML Pages

For this project, we have created multiple HTML files for different functionalities and saved them in the templates folder:

• index.html - Landing page for the application
• home.html - Main page with navigation options
• intro.html - Introduction/information page about flood prediction
• image.html - Page for image-based flood prediction input
• imageprediction.html - Page displaying image prediction results

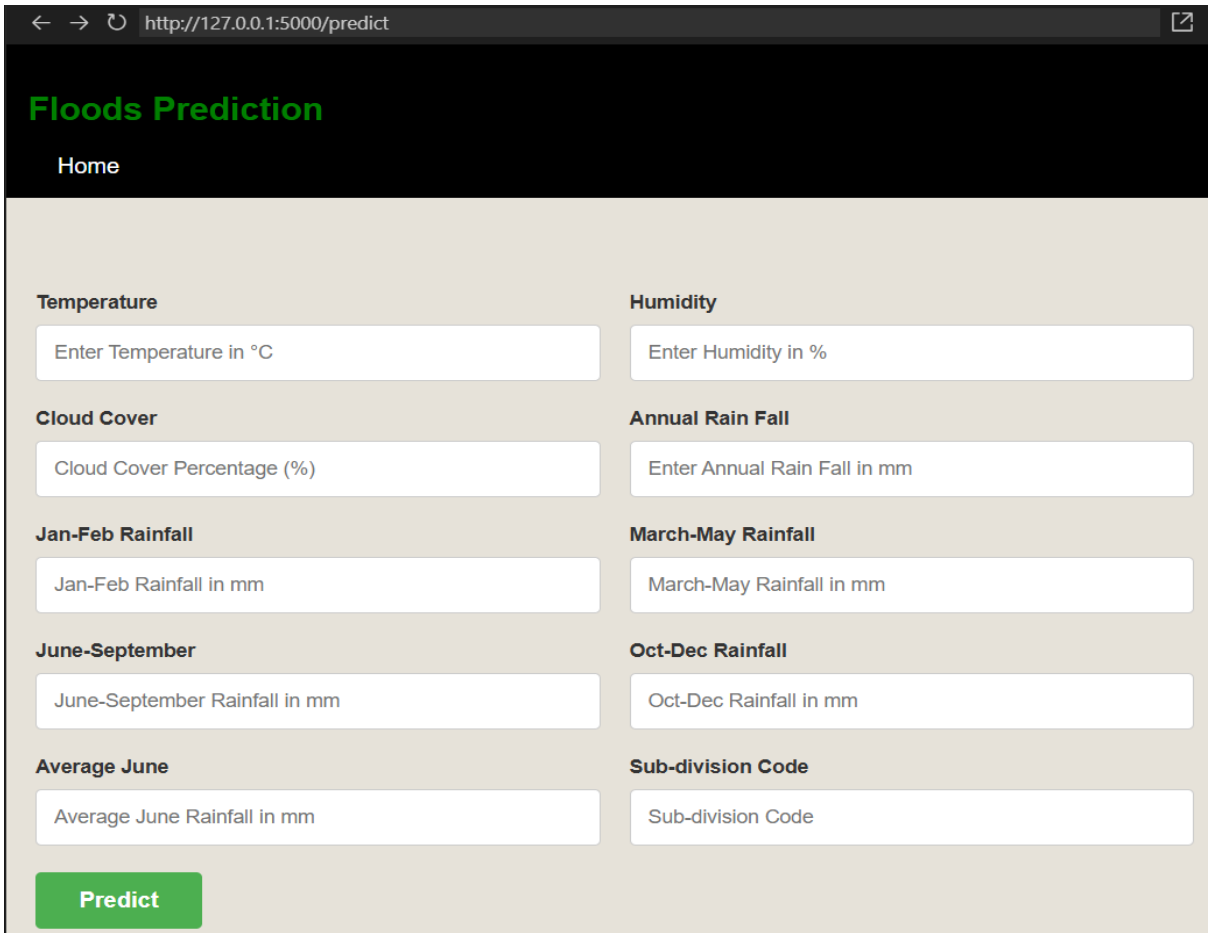
These HTML pages are styled using CSS stored in the static/css folder (styles.css) to provide an attractive and user-friendly interface.

Let's see how our main pages look:



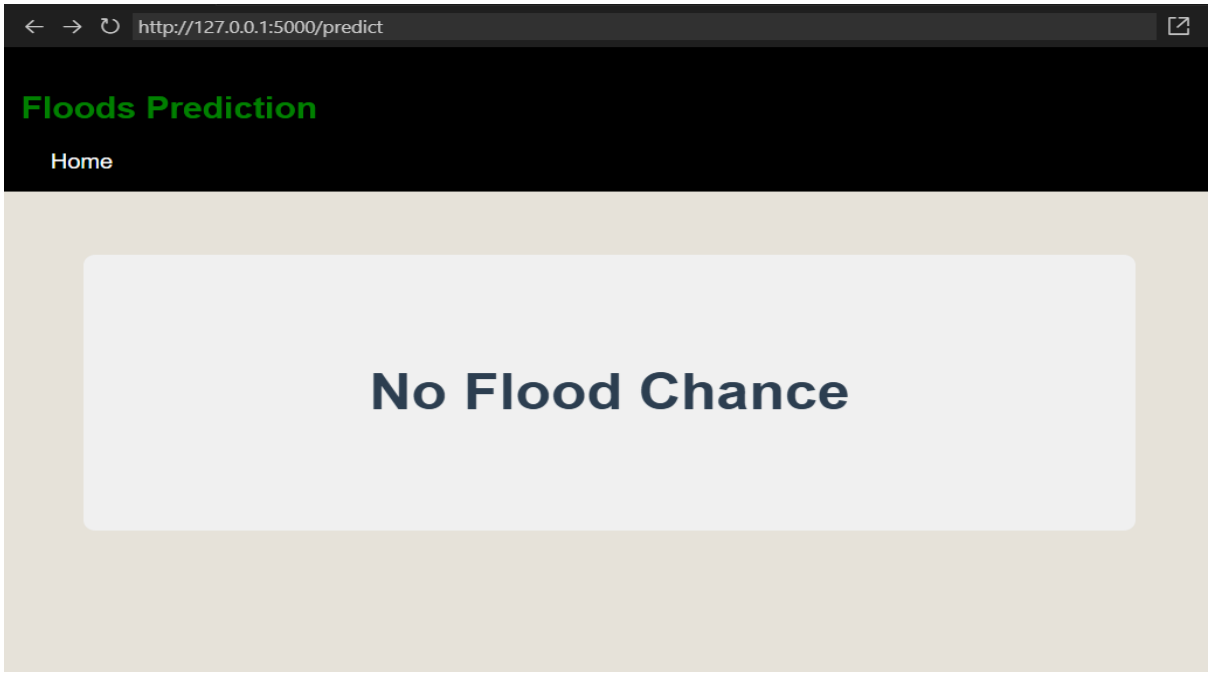
Now when you click on predict button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:



Now when you click on Submit button, you will be redirected to the prediction result page.

Let's look how our result page (imageprediction.html) looks like:



Activity 2: Build Python code:

Import the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
from flask import Flask, render_template, request
import numpy as np
from joblib import load
import os
import warnings
warnings.filterwarnings('ignore')

# Import sklearn modules BEFORE loading any models
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler
import sklearn.tree._tree

app = Flask(__name__)

# Lazy load model and scaler
base_dir = os.path.dirname(os.path.abspath(__file__))
model = None
scaler = None

def load_models():
    global model, scaler
    if model is None:
        model = load(os.path.join(base_dir, "floods.save"))
        scaler = load(os.path.join(base_dir, "transform.save"))
```

Render HTML page:

```
@app.route('/') # rendering the html template
def home():
    return render_template('home.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('/result', methods=['POST'])
def result():
    load_models()

    temp = float(request.form['temp'])
    humidity = float(request.form['humidity'])
    cloud = float(request.form['cloud'])
    annual = float(request.form['annual'])
    janfeb = float(request.form['janfeb'])
    marmay = float(request.form['marmay'])
    junsep = float(request.form['junsep'])
    octdec = float(request.form['octdec'])
    avgjune = float(request.form['avgjune'])
    sub = float(request.form['sub'])

    data = np.array([[temp, humidity, cloud, annual, janfeb, marmay, junsep, octdec, avgjune, su
    data_scaled = scaler.transform(data)

    prediction = model.predict(data_scaled)

    if prediction[0] == 1:
        return render_template('imageprediction.html', result="Flood Chance")
    else:
        return render_template('imageprediction.html', result="No Flood Chance")
```

Here we are routing our app to the result() function via the '/result' route. This function retrieves all the environmental input values (temperature, humidity, cloud cover, rainfall data, etc.) from the HTML form using POST request. These values are stored in a numpy array. Before prediction, this array is transformed using the StandardScaler (data_scaled = scaler.transform(data)) to match the training data format. The scaled data is then passed to the model.predict() function. This function returns the prediction (0 for No Flood, 1 for Flood). Based on this prediction value, the appropriate result message ("Flood Chance" or "No Flood Chance") is rendered on the imageprediction.html page.

Main Function:

```
@app.route('/result', methods=['POST'])
def result():
    load_models()

    temp = float(request.form['temp'])
    humidity = float(request.form['humidity'])
    cloud = float(request.form['cloud'])
    annual = float(request.form['annual'])
    janfeb = float(request.form['janfeb'])
    marmay = float(request.form['marmay'])
    junsep = float(request.form['junsep'])
    octdec = float(request.form['octdec'])
    avgjune = float(request.form['avgjune'])
    sub = float(request.form['sub'])

    data = np.array([[temp, humidity, cloud, annual, janfeb, marmay, junsep, octdec, avgjune, sub]])
    data_scaled = scaler.transform(data)

    prediction = model.predict(data_scaled)

    if prediction[0] == 1:
        return render_template('imageprediction.html', result="Flood Chance")
    else:
        return render_template('imageprediction.html', result="No Flood Chance")

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

Activity 3: Run the Application

• Open Command Prompt or PowerShell from the start menu
• Navigate to the Flask folder where your app.py script is located:

cd C:\Users\ponna\OneDrive\Desktop\Flood_Prediction\Flask

• Activate the virtual environment and run the application:

C:/Users/ponna/OneDrive/Desktop/Flood_Prediction/.venv/Scripts/python.exe app.py

• The Flask server will start running on http://127.0.0.1:5000 or http://localhost:5000
• Open your web browser and navigate to http://127.0.0.1:5000
• Click on the **Predict** button from the top right corner to access the prediction form
• Enter the environmental input values (temperature, humidity, rainfall data, etc.)
• Click on the **Submit** button
• View the flood prediction result on the result page