Rising Waters: A Machine Learning Approach to Flood Prediction

Project Description:

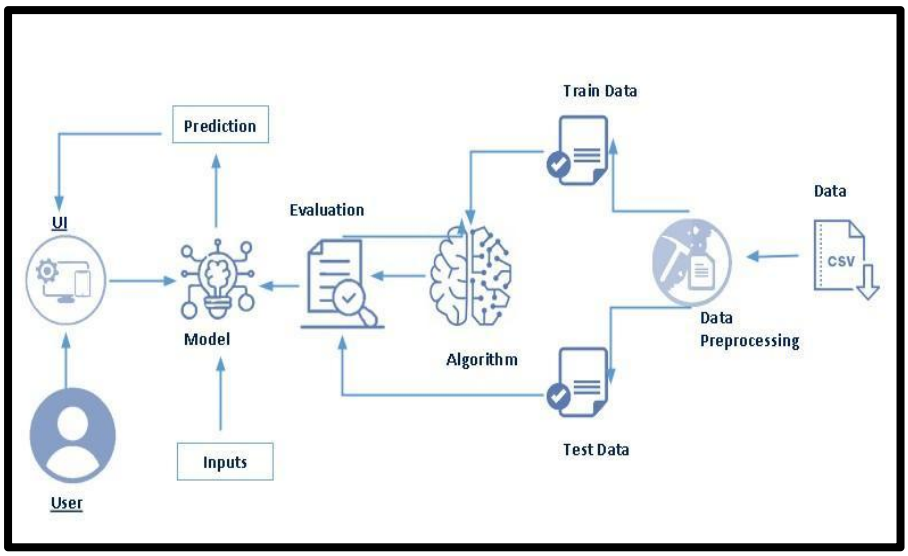
Floods are one of the most devastating natural disasters that significantly affect a country's economy, agriculture, infrastructure, and human life. Early prediction of floods is crucial for disaster management and reducing potential damage. Traditional methods for flood prediction often involve complex hydrological calculations, but they may not always provide fast and efficient results.

In this project, we develop a Machine Learning-based Flood Prediction System that analyzes environmental factors such as cloud cover and rainfall data to predict the possibility of flood occurrence. Since flood prediction depends on multiple climatic conditions, it becomes a challenging task that requires accurate data analysis and pattern recognition.

We use classification algorithms including Decision Tree, Random Forest, K-Nearest Neighbors (KNN), and XGBoost to train and test the dataset. After evaluating model performance, the best-performing model is selected and saved in `.save` (or `.pkl`) format. The trained model is then integrated with a Flask web application to provide real-time predictions through a user-friendly interface. The system can also be deployed on cloud platforms such as IBM Cloud for scalable access.

This project demonstrates the complete implementation of a Machine Learning pipeline from data preprocessing and model building to deployment for real-world flood prediction.

Technicl Architecture:



:

To complete this project, you must required 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 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:

● Understand the fundamental concepts and techniques used in Machine Learning for flood prediction.
● Gain practical knowledge in analyzing environmental and rainfall-related datasets.

● Learn data preprocessing techniques such as handling missing values, feature scaling, and train-test splitting.
● Apply classification algorithms like Decision Tree, Random Forest, KNN, and XGBoost for prediction.
● Evaluate and compare model performance to select the best-performing model.
● Deploy a trained Machine Learning model using Flask to provide real-time flood prediction through a web interface.

Project Flow:

● User interacts with the web interface to enter environmental parameters such as cloud cover and rainfall details.
● The entered input is processed and transformed using the saved scaler.
● The trained Machine Learning model analyzes the input data.
● Once the model completes analysis, the flood prediction result is displayed on the UI.

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

● Data Collection

o Collect the flood dataset containing rainfall and environmental parameters.
o Load the dataset into the system using pandas for further analysis.

● Visualizing and Analyzing Data

o Perform Univariate Analysis to understand individual features.

o Perform Bivariate Analysis to analyze relationships between features and flood occurrence.

o Perform Multivariate Analysis to study the interaction between multiple variables.

o Perform Descriptive Analysis using statistical functions like describe() and info().

● Data Pre-processing

o Check for null values and handle missing data.

o Handle outliers if present in the dataset.

o Separate independent variables (X) and dependent variable (y).

o Split the dataset into training and testing sets.

o Apply feature scaling using StandardScaler.

● Model Building

o Import model building libraries such as sklearn and xgboost.

o Initialize classification models (Decision Tree, Random Forest, KNN, XGBoost).

o Train the models using the training dataset.

o Test the models using the testing dataset.

o Evaluate model performance using appropriate metrics.

o Save the best-performing model using joblib.

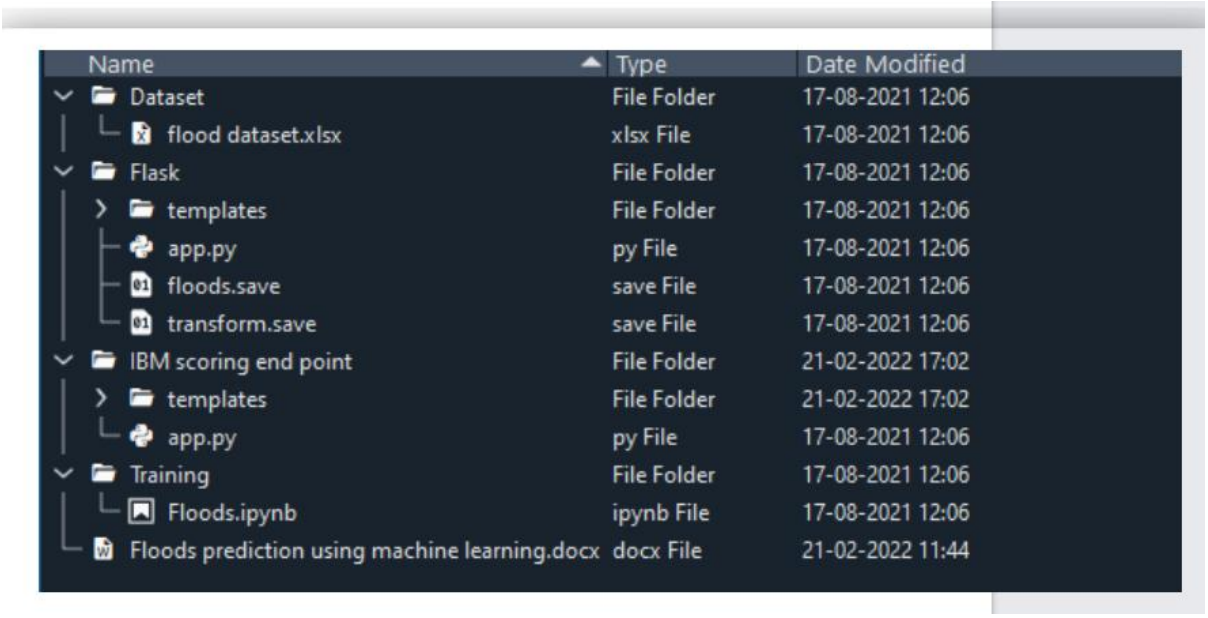
● Application Building

o Create HTML files for user interface.
o Develop Python Flask backend to integrate the trained model.
o Load the saved model and scaler inside Flask.
o Accept user input and perform prediction.
o Display the prediction result on the web page.

Project Structure:

The Flood Prediction project is organized into multiple folders to maintain clarity, modularity, and ease of deployment. The project structure separates dataset, model training, application development, and deployment components.

This structured organization ensures better maintainability and scalability of the application



Milestone 1: Data Collection

Data Collection

Machine Learning models heavily depend on data. Data is the most important component that enables model training and accurate prediction. In this project, we require a dataset containing rainfall and environmental parameters to predict flood occurrence.

Activity 1: Download the Dataset

There are many popular open-source platforms available for collecting datasets such as:

kaggle.com

UCI Machine Learning Repository

Government open data portals

In this project, we have used a Flood dataset containing parameters such as:

Cloud Cover

Annual Rainfall

Seasonal Rainfall Data

Flood Occurrence (Target Variable)

The dataset is downloaded and stored inside the Dataset folder for further processing.

Link:flood dataset.xlsx

Milestone 2: Visualizing and analysing the data

After downloading the dataset, we analyze and understand the data using visualization and statistical techniques.

Note: There are many techniques available for data analysis. In this project, we have used essential visualization and descriptive methods.

Activity 1: Importing the libraries

We import necessary libraries such as:

numpy

pandas

matplotlib

Seaborn

sklearn

These libraries help in data manipulation, visualization, and model building.

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import joblib
```

Activity 2: Read the Dataset

The dataset may be in formats such as .csv, .xlsx, .txt, or .json.

Using pandas, we read the dataset:

read_csv() for CSV files

read_excel() for Excel files

After loading, we examine:

df.head()

df.shape

```python
dataset = pd.read_excel('../Dataset/flood dataset.xlsx')
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

```python
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
```

Activity 3: Univariate analysis

Univariate analysis means analyzing a single feature at a time.

In this project, we use:

Histograms

Distplots

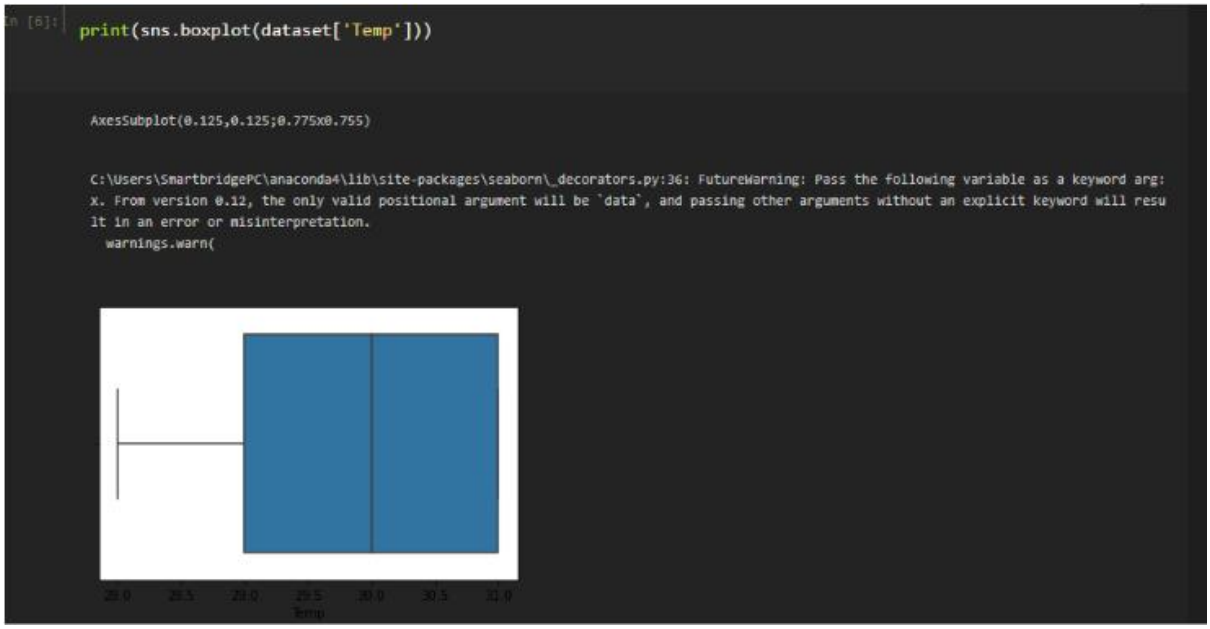
Countplots

This helps us understand:

Distribution of rainfall values

Frequency of flood occurrences

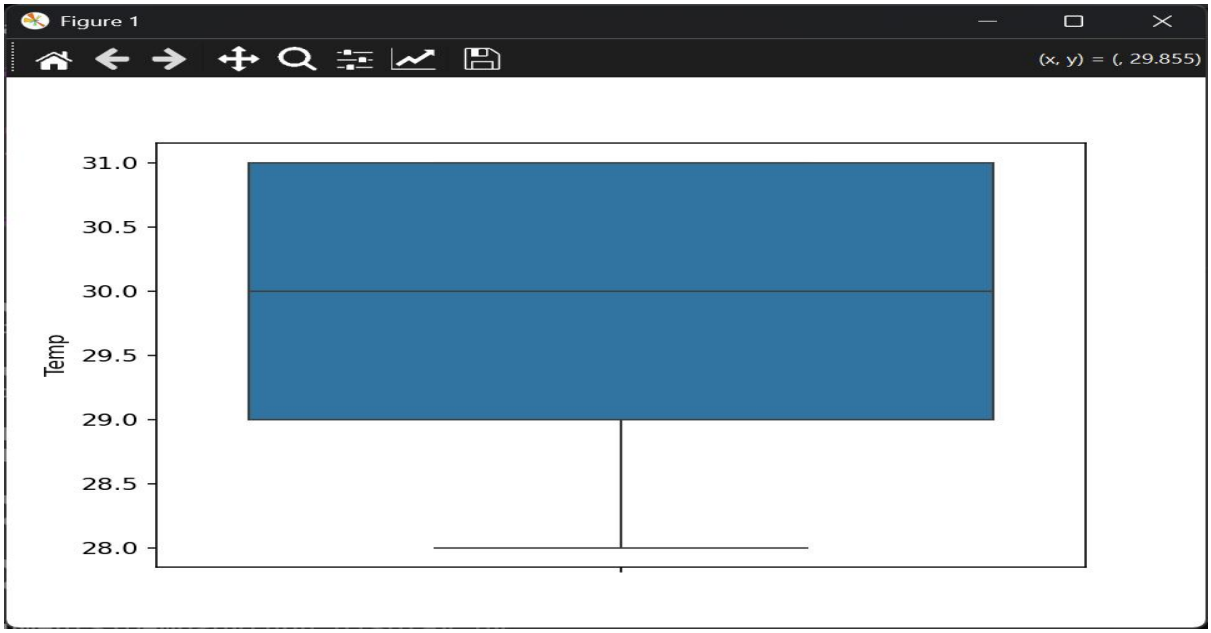
Data skewness



- 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

Activity 4:Bivariate analysis



Bivariate analysis helps us understand the relationship between two variables.

Examples:

Rainfall vs Flood occurrence

Cloud Cover vs Flood occurrence

We use countplot and scatter plots to analyze how features influence the target variable.

display.py X flood dataset.xlsx

display.py > ...

```
1   import numpy as np
2   import pandas as pd
3   import matplotlib.pyplot as plt
4   import seaborn as sns
5   dataset = pd.read_excel("Dataset\\flood dataset.xlsx")
6
7   sns.displot(dataset['Temp'])
8   plt.show()
9
10  sns.boxplot(dataset['Temp'])
11  plt.show()
```

Activity 5: Multivariate analysis

Multivariate analysis studies relationships among multiple variables simultaneously.

In this project, we analyze how different rainfall parameters together impact flood occurrence.

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)
```

Activity 6: Descriptive analysis

Descriptive analysis helps summarize dataset statistics.

Using:

df.describe()

df.info()

df.isnull().sum()

We understand:

Mean

Standard deviation

Minimum and maximum values

Null values

To check the first five rows of the dataset, we have a function called **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

Text data type is known a 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.

```
dataset.isnull().any()
dataset.info()
dataset.describe().T
```

[15] ✓ 0.0s Python

```
<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
```

	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

Milestone 3: Data Pre-processing

Raw data is not directly suitable for model training. It may contain inconsistencies, missing values, or scale differences.

Data preprocessing includes:

Handling missing values

Encoding categorical data (if present)

Feature scaling

Splitting dataset

Activity 1: Checking for null values

We check:

df.shape()

df.info()

df.isnull().sum()

If missing values are found, they are handled using:

Mean (for numerical features)

Mode (for categorical features)

```
dataset.isnull().any()
# dataset.info()
# dataset.describe().T
```

✓ 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
```

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

Activity 2: Handling Categorical Values

If categorical features exist, they are converted into numerical form using encoding techniques such as:

- Manual encoding
- Label encoding

Activity 3: Feature Scaling

Scaling ensures that all features are in similar ranges.

We use:

StandardScaler()

Scaling is especially important for distance-based algorithms like KNN.

Activity 4: Scaling the Data

We separate dataset into:

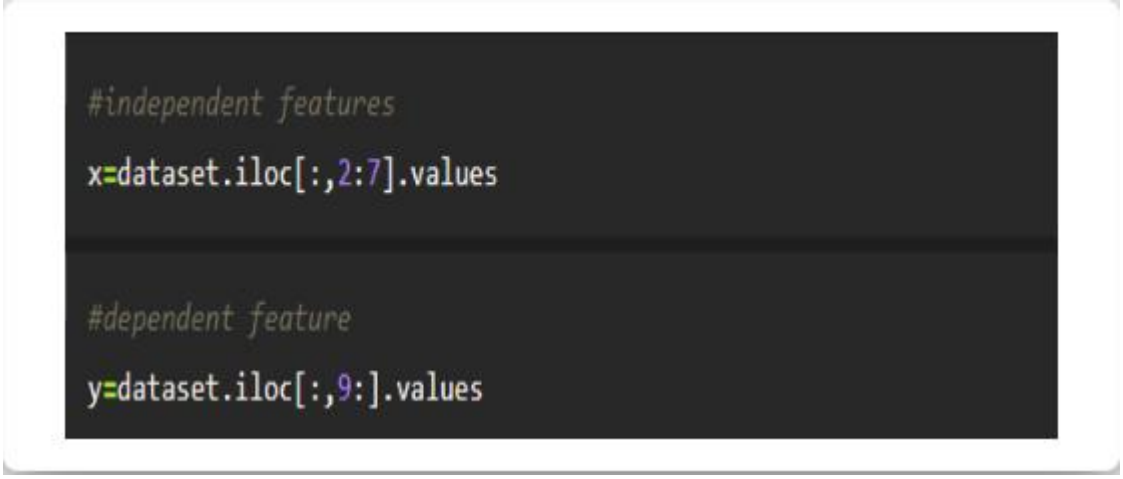
X → Independent variables

y → Target variable

Using:

train_test_split()

The dataset is split into training and testing sets to evaluate model performance.



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.

Activity 5 : 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)
```

```
#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)
```

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 multiple classification models. The best model is saved based on its performance.

Activity 1: Decision tree model

- Initialize DecisionTreeClassifier
- Train using `.fit()`
- Predict using `.predict()`
- Evaluate using accuracy and confusion matrix.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor

dtree = DecisionTreeRegressor()
Rf = RandomForestRegressor()
knn = KNeighborsRegressor()
xgb = XGBRegressor()
```

✓ 0.0s Python

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.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor

dtree = DecisionTreeRegressor()
Rf = RandomForestRegressor()
knn = KNeighborsRegressor()
xgb = XGBRegressor()
```

✓ 0.0s Python

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.

```
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)
```

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.

```
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)
```

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.

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 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 , 89.1% 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().

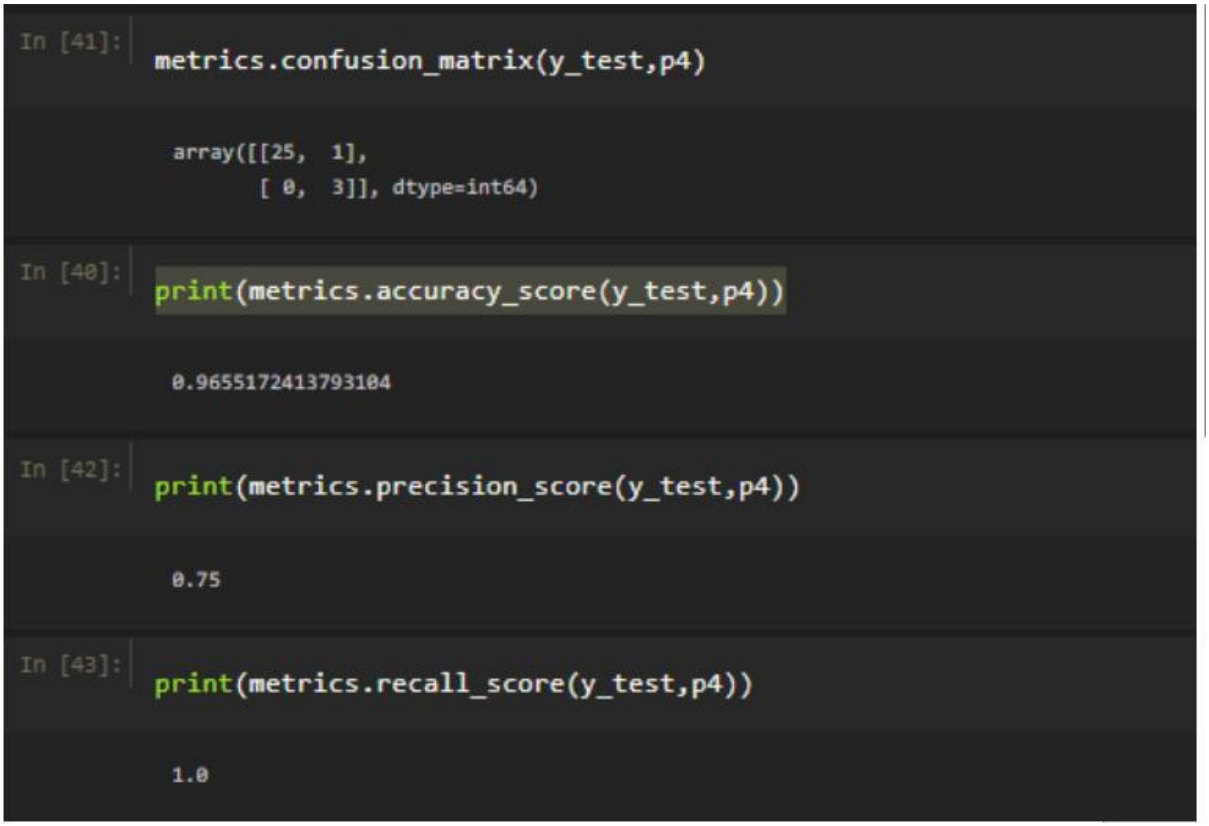
The best-performing model is saved using:

joblib.dump()

The scaler is also saved for consistent prediction during deployment.

- 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.

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



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 server-side script using Flask

Activity1: Building Html Pages:

We create the following HTML files:

home.html

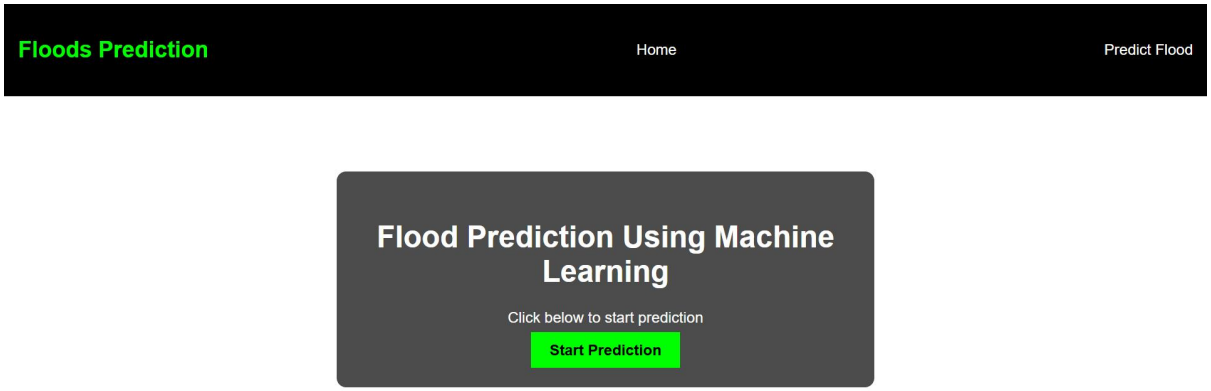
index.html

chance.html

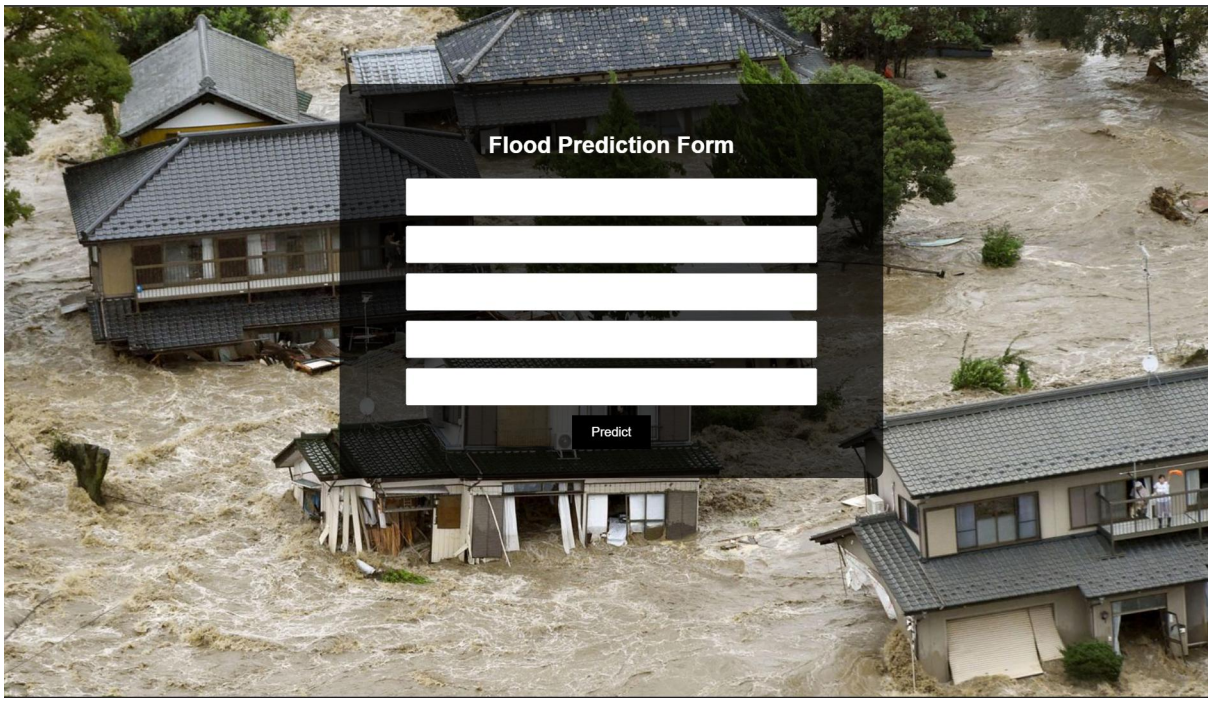
noChance.html

These files are saved inside the 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 predict.html. Lets look how our predict.html file looks like:



Now when you click on submit button from left bottom corner you will get redirected to submit.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
from joblib import load

# Create Flask app
app = Flask(__name__)

# Load saved model and scaler
model = load('floods.save')
sc = load('transform.save')
```

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.

- 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 = Flask(__name__)

# Load saved model and scaler
model = load('floods.save')
sc = load('transform.save')
```

Render HTML page:

```
@app.route('/')
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 be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict')
def predict_page():
    return render_template('index.html')

@app.route('/data_predict', methods=['POST'])
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 == 1:
        result = "Possibility of Severe Flood"
    else:
        result = "No Possibility of Severe Flood"

    return render_template("chance.html", prediction=result)


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

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:

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

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.

```
(base) C:\Users\kales\Documents\Project\Flask>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
```