

Rising Waters: A Machine Learning Approach to Flood Prediction

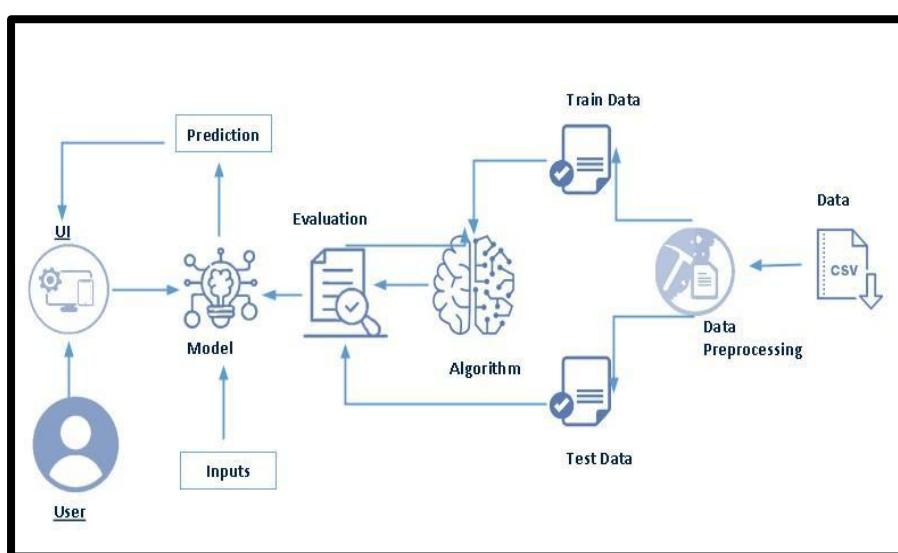
Project Description:

Floods are among the most destructive natural disasters, causing major damage to infrastructure, agriculture, and human life. Due to changing climate patterns and unpredictable rainfall, early flood prediction has become essential for disaster management. Traditional forecasting methods are often not sufficient, making machine learning-based systems important for improving prediction accuracy and providing timely warnings.

Flood prediction is challenging because it depends on multiple environmental factors such as rainfall, temperature, humidity, and water levels. Analyzing historical climate data helps identify patterns that indicate flood risk. Early detection allows authorities and communities to take preventive actions and reduce potential losses.

In this project, classification algorithms such as Decision Tree, Random Forest, SVM, XGBoost, and Extra Trees Classifier are trained using environmental features. After performance evaluation, the best-performing model (XGBoost) is selected and integrated into a Flask-based web application, where users can enter environmental details and instantly receive a prediction indicating whether flood risk is High or Low.

Technical Architecture:



Pre requisites:

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>
 - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
 - Metrics: <https://youtu.be/aWAnNHXIKww>
- **Flask Basics :** https://www.youtube.com/watch?v=lj4l_CvBnt0

Project Objectives:

By the end of this project you will:

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

Project Flow:

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

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

- Data collection
 - Collect the dataset or create the dataset
- Visualizing and analyzing data
 - Univariate analysis
 - Bivariate analysis
 - Multivariate analysis
 - Descriptive analysis
- Data pre-processing
 - Checking for null values
 - Handling outlier
 - Handling categorical data
 - Splitting dataset into Dependent And Independent Variables
 - Splitting data into train and test
 - Feature Scaling
- 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

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

Milestone 1: Data Collection

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

Activity 1: Download the dataset

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

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

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

Milestone 2: Visualizing and analysing the data

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

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1: Importing the libraries

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

Importing Libraries¶

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
```

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.

The screenshot shows a Jupyter Notebook cell with the title "LOADING DATASET". The code cell contains:

```
df = pd.read_excel("Flood dataset.xlsx")
```

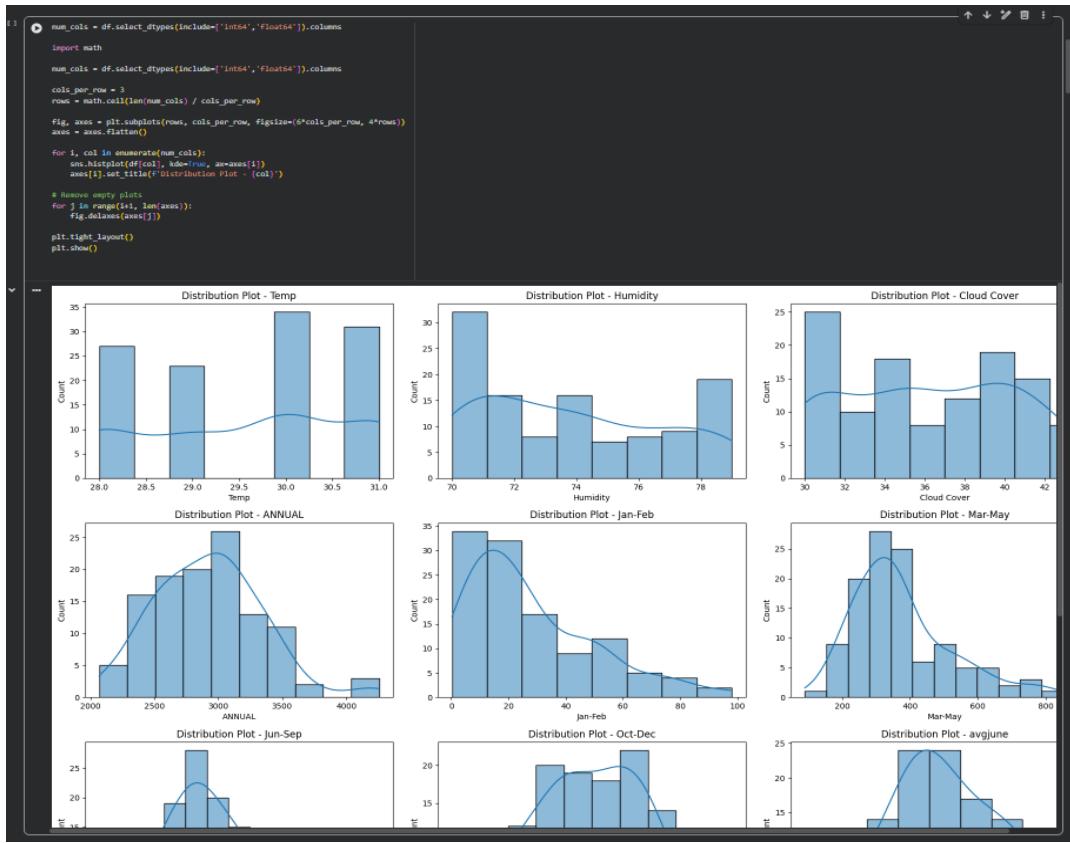
Below the code cell, the output shows the first 10 rows of the dataset:

	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
5	30	70	38	2708.0	34.1	230.0	1943.1	500.8	138.300000	254.1	0
6	29	74	40	3671.1	23.7	328.0	2737.8	581.7	256.966667	669.5	1
7	30	78	36	2648.3	28.8	283.7	2023.6	312.2	197.533333	450.0	0
8	30	71	40	3050.2	65.9	628.3	1940.4	415.5	234.900000	231.5	0
9	30	70	34	2848.6	28.4	296.7	1886.5	637.0	228.666667	531.2	0

Activity 3: Univariate analysis

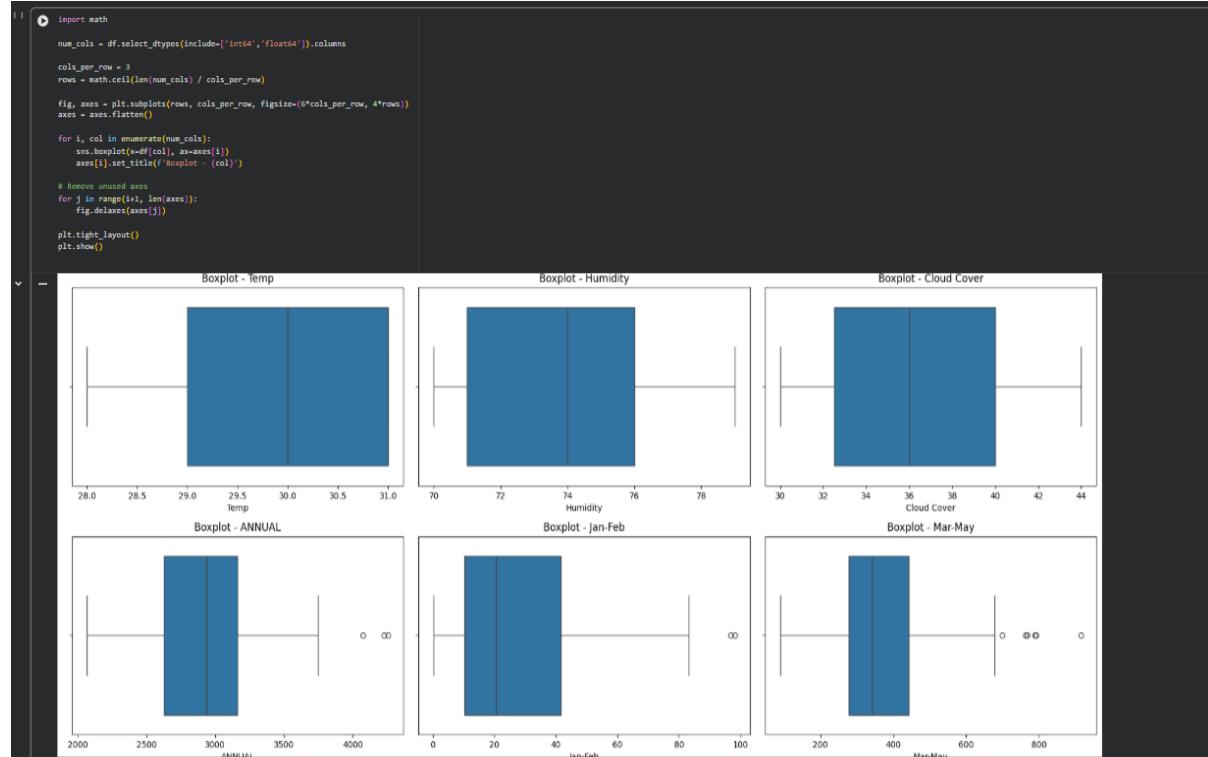
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

- 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`.



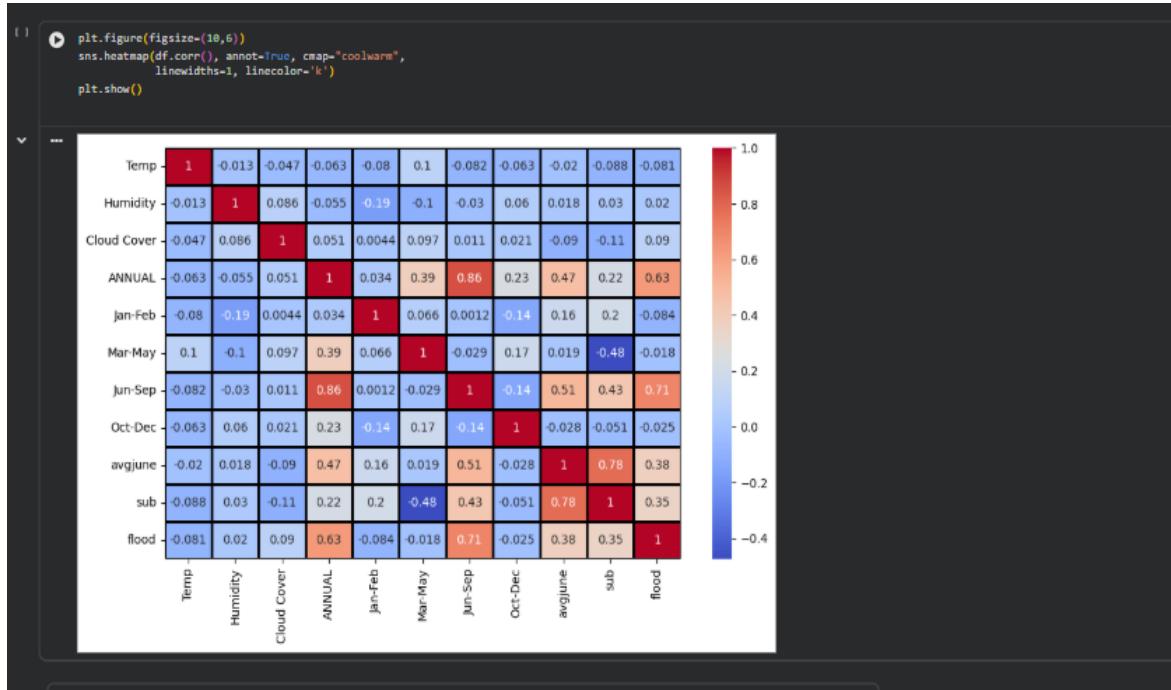
From the above graph, we can infer that the column temp follows some kind of normal distribution, it means the data is normal is almost normal

Box Plots:



Activity 4: Multivariate analysis

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



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.

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

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.info()
df.describe()

<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

```

	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	38.286957	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	22.381032	151.091850	388.254397	129.88043	62.547597	210.438813	0.347597
min	28.000000	70.000000	30.000000	2088.800000	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	1788.850000	407.450000	179.666667	295.000000	0.000000
50%	30.000000	74.000000	38.000000	2937.500000	20.500000	342.200000	1948.700000	501.500000	211.033333	430.800000	0.000000
75%	31.000000	78.000000	40.000000	3184.100000	41.800000	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	386.068667	982.700000	1.000000

This is formatted as code

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.

Milestone 3: Data Pre-processing

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

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Splitting dataset into Dependent And Independent Variables
- Splitting data into train and test
- 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.

Activity 1: Checking for null values

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

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

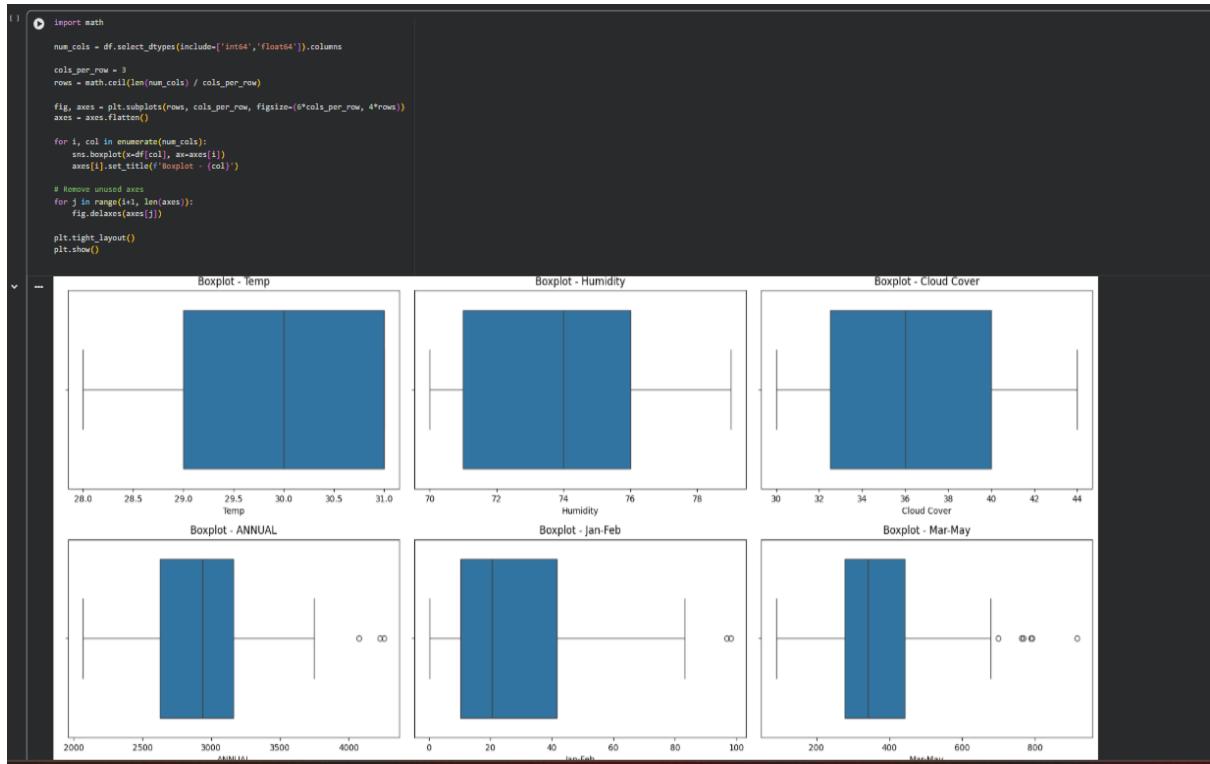
Temp	False
Humidity	False
Cloud Cover	False
ANNUAL	False
Jan-Feb	False
Mar-May	False
Jun-Sep	False
Oct-Dec	False
avgjune	False
sub	False
flood	False
dtype:	bool

From the above code of analysis, we can infer that columns such as `newbalanceOrg`, `oldbalanceOrig`, `isFraud` are having the missing values, we need to treat them in a required way.

Activity 2: 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.

Activity 3: Handeling 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.

Activity 4: Splitting 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 our independent and dependent variables:**

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

#dependent feature
y=dataset.iloc[:,9:1].values
```

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

```
#split the data into train and test set from our x and y
import train_test_split function
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)
```

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`.

Activity 6: 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.

```
FEATURE SCALING

[1]: scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)

Saving the standard Scaler

[1]: import joblib
      joblib.dump(scaler, "transform.save")
      ['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. 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.

```
DECISION TREE

[1]: dt = DecisionTreeClassifier(random_state=42)
      dt.fit(X_train, y_train)

      + DecisionTreeClassifier
      DecisionTreeClassifier(random_state=42)
```

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.

RANDOM FOREST

```
[ ] rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

+

```
+ RandomForestClassifier ⓘ ⓘ
RandomForestClassifier(random_state=42)
```

This screenshot shows a Jupyter Notebook cell for a 'RANDOM FOREST' model. The code defines a 'rf' variable as a 'RandomForestClassifier' with 100 estimators and a random state of 42. It then fits this model to training data 'X_train' and 'y_train'. Below the code, the output shows the creation of a 'RandomForestClassifier' object with the specified parameters.

Activity 3: KNN

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

KNN

```
[ ] knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

+

```
+ KNeighborsClassifier ⓘ ⓘ
KNeighborsClassifier()
```

This screenshot shows a Jupyter Notebook cell for a 'KNN' model. The code defines a 'knn' variable as a 'KNeighborsClassifier' with 5 neighbors. It then fits this model to training data 'X_train' and 'y_train'. Below the code, the output shows the creation of a 'KNeighborsClassifier' object with the specified parameters.

Activity 5: 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.

XG BOOST

```
[ ] xg_clas = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=4,
    random_state=42
)
xg_clas.fit(X_train, y_train)
```

...

```
+ XGBClassifier ⓘ ⓘ
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=4,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=100,
              n_jobs=None, num_parallel_tree=None, ...)
```

This screenshot shows a Jupyter Notebook cell for an 'XG BOOST' model. The code defines an 'xg_clas' variable as an 'XGBClassifier' with 100 estimators, a learning rate of 0.1, a maximum depth of 4, and a random state of 42. It then fits this model to training data 'X_train' and 'y_train'. Below the code, the output shows the creation of an 'XGBClassifier' object with the specified parameters.

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

Activity 5: Compare the model

```
Comparing the models
[1]: print("KNN:", accuracy_score(y_test,p1 ))
      print("Decision Tree:", accuracy_score(y_test,p2 ))
      print("Random Forest:", accuracy_score(y_test,p3 ))
      print("Xg Boost:", accuracy_score(y_test,p4 ))

...  KNN: 0.896551724137931
      Decision Tree: 0.9655172413793104
      Random Forest: 0.9655172413793104
      Xg Boost: 0.9655172413793104
```

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 96.5%.so we considering xgboost and deploying this model.

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

```
from sklearn import metrics
import pandas as pd

results = []

models = {
    "KNN": p1,
    "Decision Tree": p2,
    "Random Forest": p3,
    "XGBoost": p4
}

for name, pred in models.items():
    acc = metrics.accuracy_score(y_test, pred)
    prec = metrics.precision_score(y_test, pred)
    rec = metrics.recall_score(y_test, pred)
    f1 = metrics.f1_score(y_test, pred)

    results.append([name, acc, prec, rec, f1])

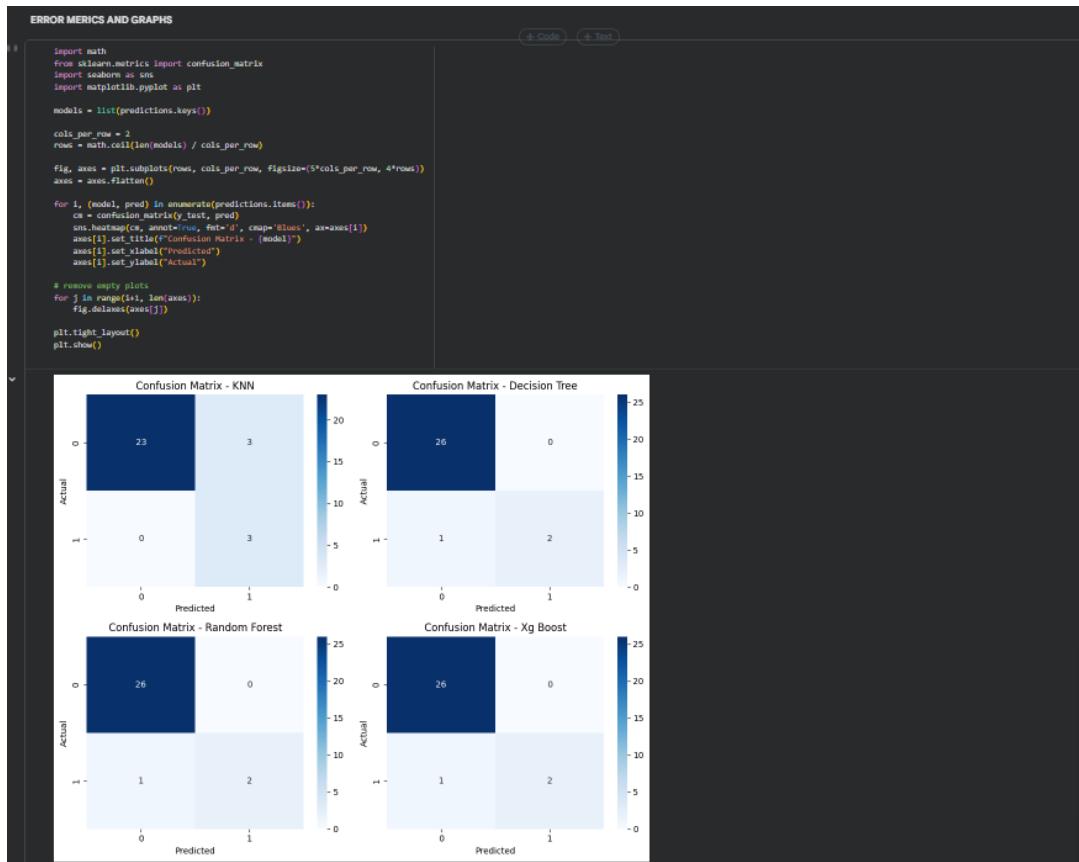
results_df = pd.DataFrame(results,
                          columns=["Model", "Accuracy", "Precision", "Recall", "F1-Score"])

print(results_df)
```

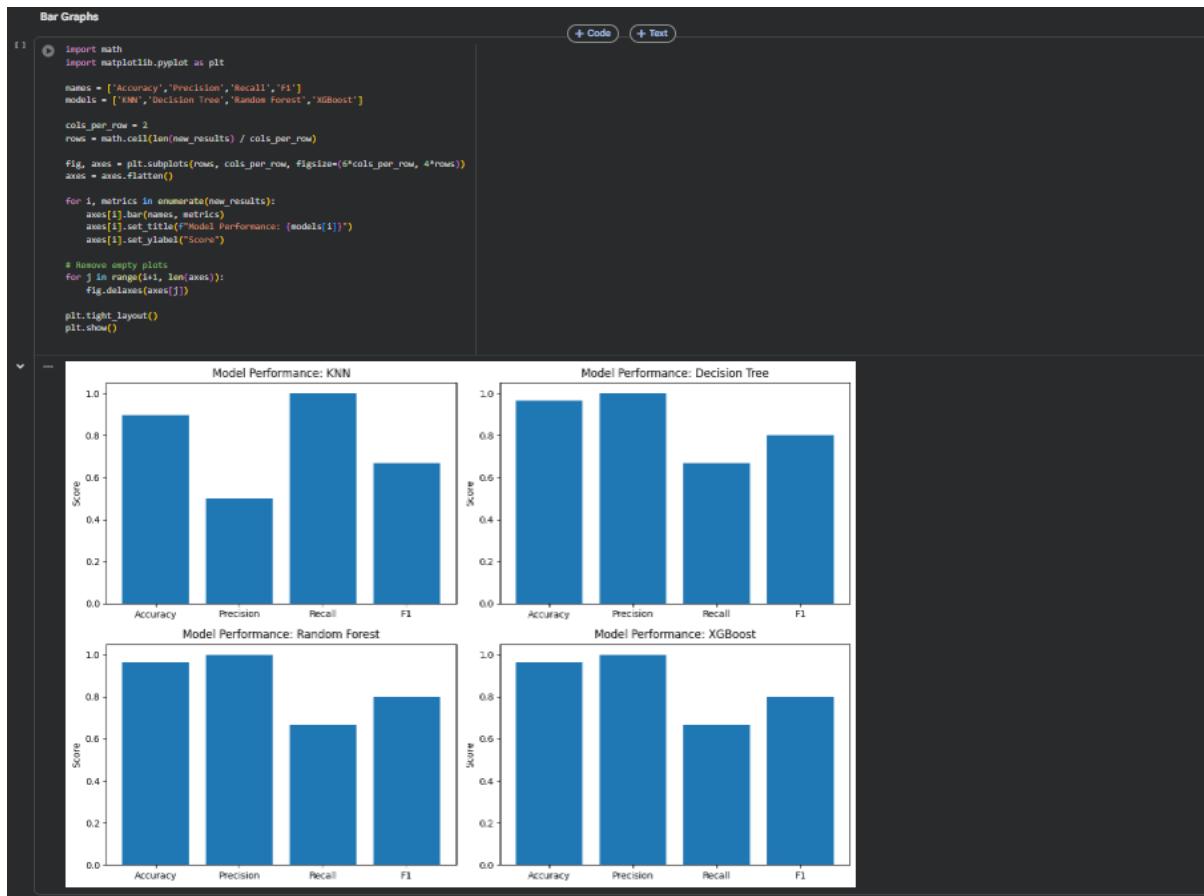
	Model	Accuracy	Precision	Recall	F1-Score
0	KNN	0.896552	0.5	1.000000	0.666667
1	Decision Tree	0.965517	1.0	0.666667	0.800000
2	Random Forest	0.965517	1.0	0.666667	0.800000
3	XGBoost	0.965517	1.0	0.666667	0.800000

Prefomance evaluation graphs:

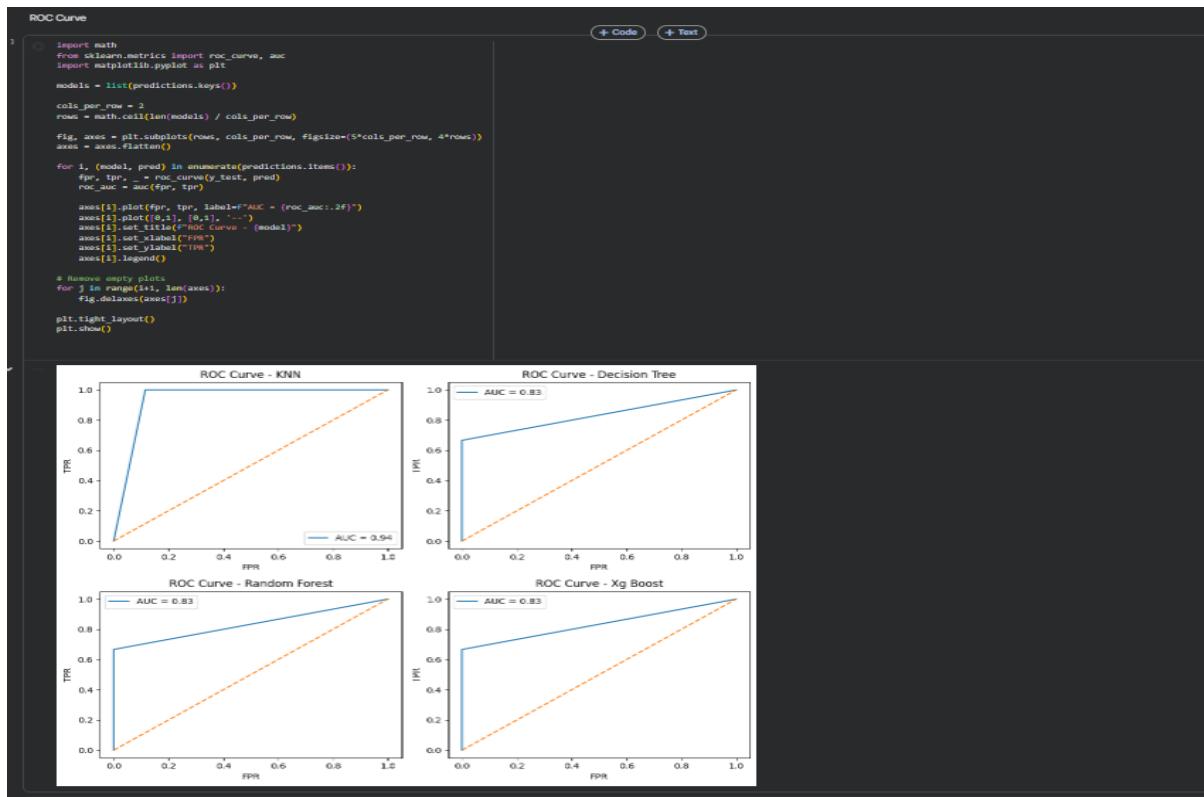
Confusion Metrics



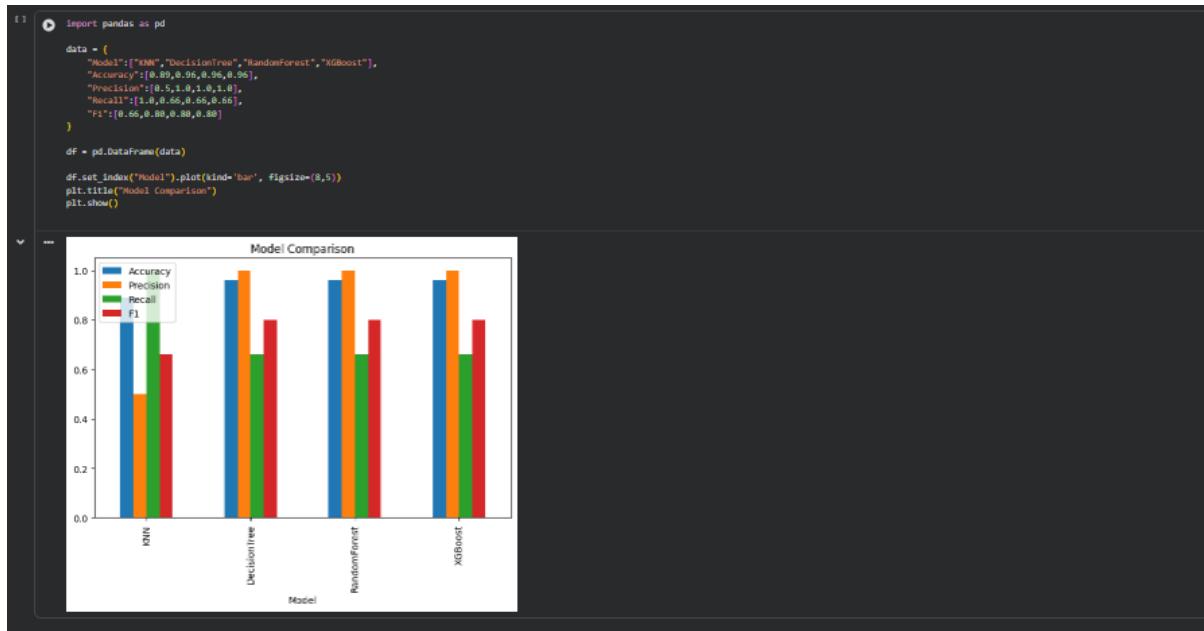
Bar Graphs:



Roc Curves:



Models Comparison Graph:



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

```

BEST-FIT XgBoost

SAVING XGBoost

[ ] import joblib
joblib.dump(xg_clas, "xg_flood_model.pkl")

['xg_flood_model.pkl']

[ ] joblib.dump(features, "model_columns.pkl")
... ['model_columns.pkl']

Save Scaler Separately

[ ] joblib.dump(scaler, "xg_scaler.pkl")

['xg_scaler.pkl']

```

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 user where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

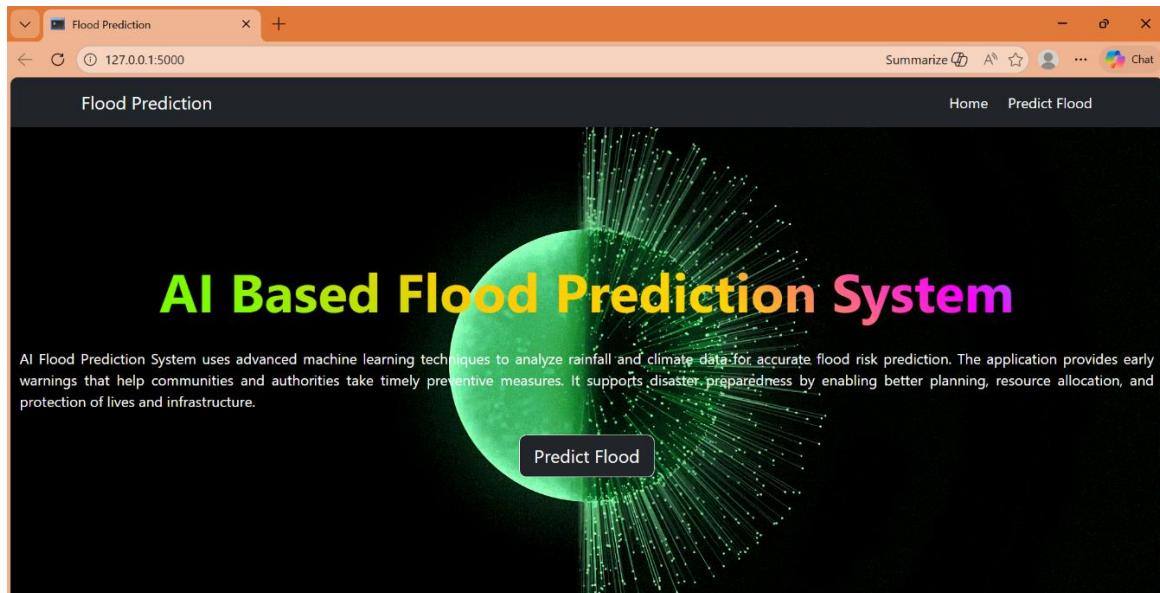
Activity1: Building Html Pages:

For this project create three HTML files namely

- index.html
- predict.html
- base.html

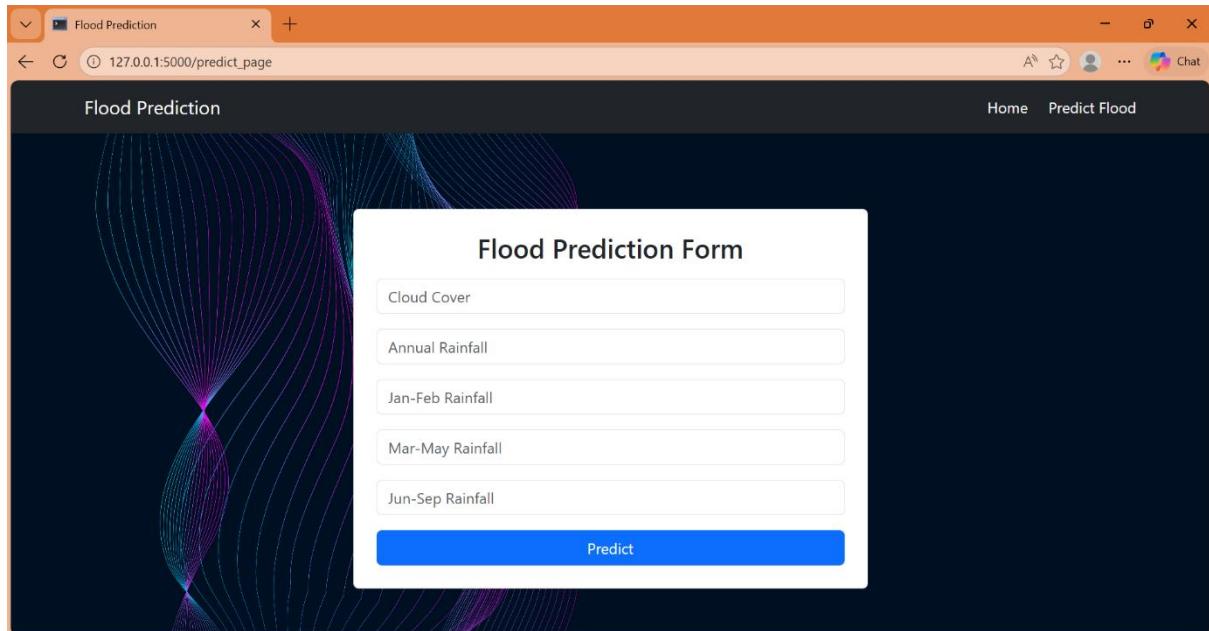
and save them in templates folder.

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



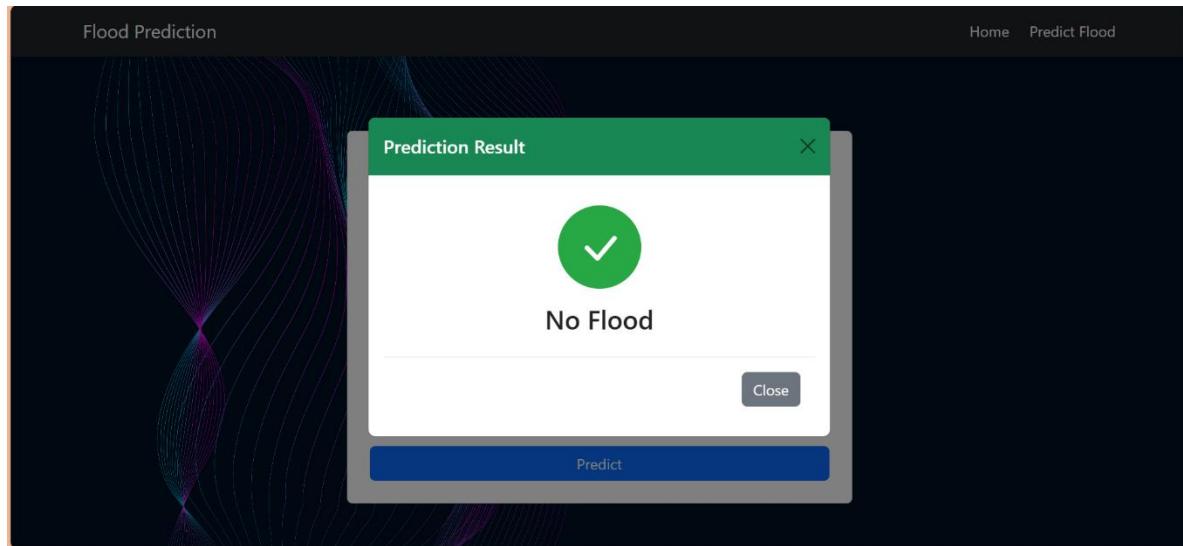
Now when you click on \Predict Flood button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:



Now when you click on detect button a popup will shown weather the Flood is Expected or Not based on the entered Data.

Lets look how our Pop up looks like:



Activity 2: Build Python code:

Import the libraries

```
1  from flask import Flask, render_template, request  
2  import numpy as np  
3  import joblib
```

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 declared constructor to route to the HTML page which we have created earlier.

In the example, '/' URL is bound with index.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:

```
        bhatt C:\Users\bhatt          4
        > ⌂ .anaconda      5     app = Flask(__name__)
        > ⌂ .cache         6
        > ⌂ .conda         7     model = joblib.load("xg_flood_model.pkl")
        > ⌂ .config        8     xg_scaler = joblib.load("xg_scaler.pkl")
        > ⌂ .continuum     9
        > ⌂ .idlerc       10
        > ⌂ .ipynb_checkpoints 11     @app.route('/')
        > ⌂ .ipython       12     def home():
        > ⌂ .jupyter      13             return render_template('home.html')
        > ⌂ .keras         14
        > ⌂ .matplotlib    15     @app.route('/predict_page')
        > ⌂ .VirtualBox   16     def predict_page():
        > ⌂ .vscode        17             return render_template('predict.html')
        > ⌂ anaconda_projects 18
        > ⌂ AppData        19     @app.route(rule: '/predict', methods=['POST'])
        > ⌂ Application Data 20     def predict():
        > ⌂ Contacts       21             data = [
        > ⌂ Cookies        22                 float(request.form['cloud']),
        > ⌂ CrossDevice    23                 float(request.form['annual']),
        > ⌂ Documents       24                 float(request.form['janfeb']),
        > ⌂ Downloads       25                 float(request.form['marchmay']),
        > ⌂ Favorites       26                 float(request.form['junsep'])
        > ⌂ 
```

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 be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```

> .anaconda
> .cache
> .conda
> .config
> .continuum
> .idlerc
> .ipynb_checkpoints
> .ipython
> .jupyter
> .keras
> .matplotlib
> .VirtualBox
> .vscode
> anaconda_projects
> AppData
  Application Data
> Contacts
  Cookies

```

```

21     data = [
22         float(request.form['cloud']),
23         float(request.form['annual']),
24         float(request.form['janfeb']),
25         float(request.form['marchmay']),
26         float(request.form['junsep'])
27     ]
28
29     df = np.array([data])
30     scaled = xg_scaler.transform(df)
31     pred = model.predict(scaled)[0]
32
33     result = "Flood Expected" if pred == 1 else "No Flood"
34
35     return render_template('predict.html', prediction=result)
36
37 ▶  if __name__ == "__main__":
38     app.run(debug=True)
39

```

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.

