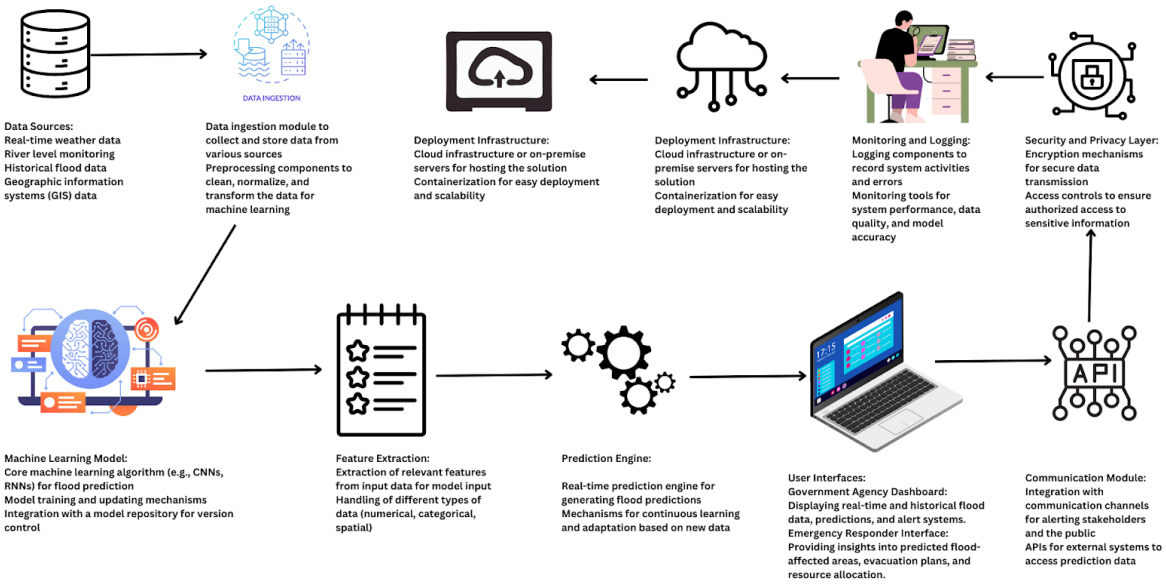
Introduction:

Floods, with their inevitable occurrence, inflict severe consequences, leading to loss of lives, displacement, and inadequate post-flood assistance. The critical issue of delayed alerts in flood-prone areas compounds the challenges, highlighting a significant loophole in disaster management. Conventional systems often fall short in providing timely forecasts, amplifying the vulnerability of economies to such natural disasters.

Recognizing the urgency to address this issue, our project leverages machine learning to enhance flood prediction accuracy. By analyzing historical weather data from specific regions, a predictive model is developed using various machine learning algorithms, including Decision Trees, Random Forest, KNN, and XGBoost. The resulting model is integrated into a web application accessible to the relevant authorities for efficient monitoring.

Through the application of classification algorithms, we aim to train and test the data meticulously. The best-performing model is then selected, saved in a pickle (pkl) format, and further integrated into a web framework using Flask. Additionally, we implement IBM deployment to ensure the scalability and accessibility of the flood prediction system, emphasizing the project's commitment to proactive disaster management and community well-being.

Technical Architecture:



Pre-Requisites:

Anaconda navigator
Numpy
Pandas
Scikit-learn

Matplotlib
Pickle-mixin
Seaborn
Flask

Project Objective:

The primary objective of the "Rising Waters" project is to leverage machine learning techniques to enhance flood prediction accuracy, providing timely alerts and minimizing the impact of devastating floods. The project aims to address the critical issues associated with conventional flood prediction systems, which often exhibit delays in forecasting floods in flood-prone areas.

The specific goals of the project are as follows:

Machine Learning-Based Flood Prediction: Implement machine learning algorithms to analyze historical weather data for specific regions. Develop predictive models capable of accurately forecasting floods, considering various factors such as rainfall, river levels, soil moisture, and other relevant parameters.

Timely Alerts: Create a robust system that generates timely alerts based on the predictions made by the machine learning models. The goal is to bridge the gap in current flood alert systems, ensuring that authorities and residents receive advance notice to take necessary actions and minimize the impact of floods on human lives and infrastructure.

Web Application Integration: Develop a user-friendly web application that allows concerned authorities to monitor and access the flood prediction system. The web application should provide real-time updates, visualizations, and detailed information about the predicted flood occurrences for different regions.

Accuracy Improvement: Continuously refine and optimize the machine learning models to enhance prediction accuracy over time. Incorporate feedback mechanisms and additional data sources to improve the reliability of the flood prediction system.

Community Outreach: Educate and raise awareness within the communities residing in flood-prone areas. Provide them with accessible information through the web application, empowering them to take informed actions in response to flood alerts.

Flask Integration: Implement Flask, a web framework for Python, to seamlessly integrate the machine learning models with the web application. Ensure smooth communication and real-time updates between the backend prediction system and the frontend interface.

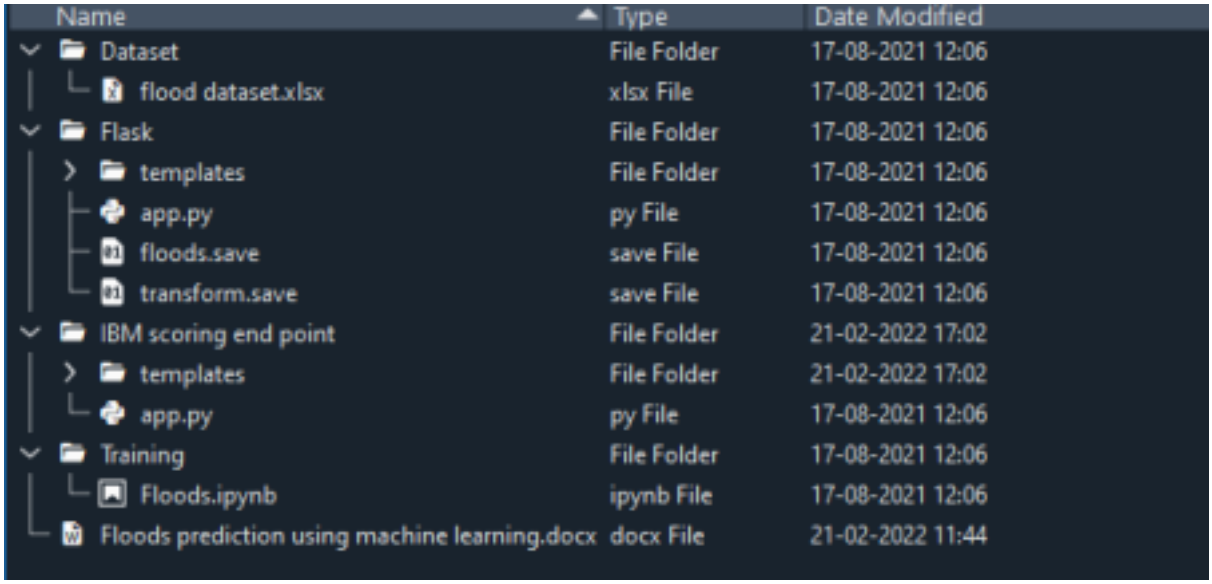
Scalability and Robustness: Design the system with scalability and robustness in mind, allowing it to handle large datasets and adapt to evolving weather patterns. Ensure the system's resilience to handle increased loads during critical situations.

Project Flow:

1. Install Required Libraries.
2. Data Collection.
   Collect the dataset or Create the dataset
3. Data Preprocessing.
   Import the Libraries.
   Importing the dataset.
   Understanding Data Type and Summary of features.
   Take care of missing data
   Data Visualization.
   Drop the column from DataFrame & replace the missing value.
   Splitting the Dataset into Dependent and Independent variables
   Splitting Data into Train and Test.

4. Model Building
   Training and testing the model
   Evaluation of Model
   Saving the Model
5. Application Building
   Create an HTML file
   Build a Python Code
6. Final UI
   Dashboard Of the flask app.

Project Structure:



Milestone 1: Data Collection:
https://www.kaggle.com/datasets/arbethi/rainfall-dataset

Milestone 2: Visualizing and analyzing the data

Activity 1: Importing the libraries

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

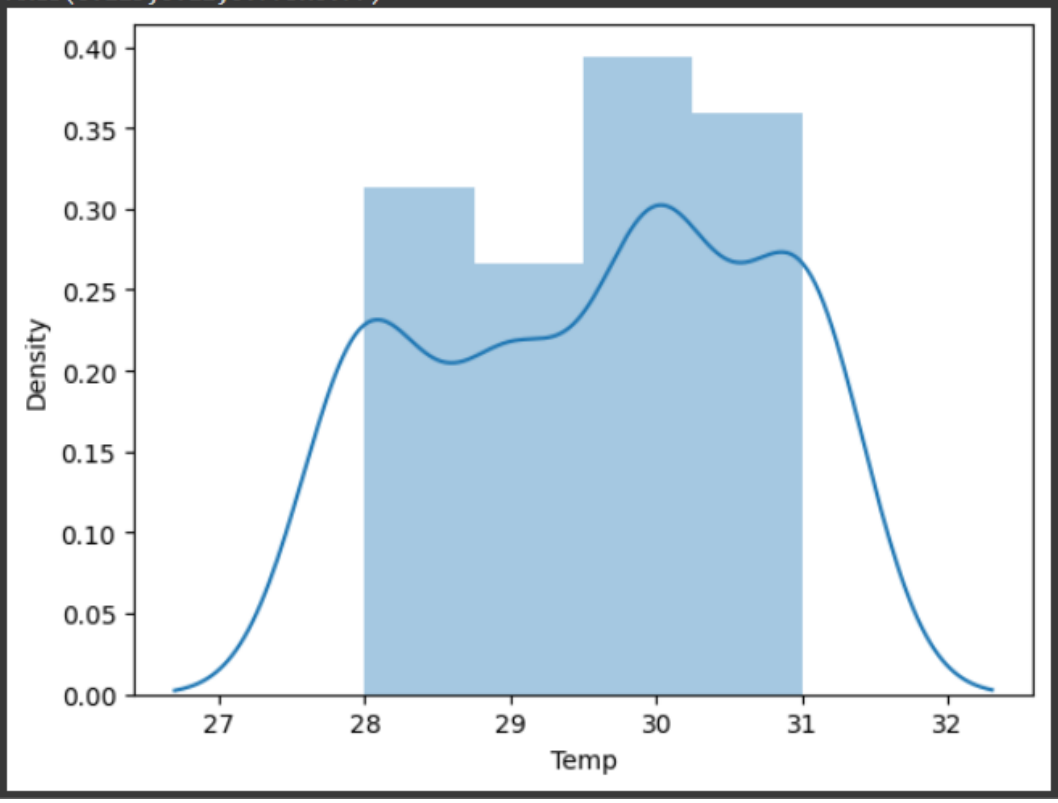
Activity 2: Reading the Dataset

```
data=pd.read_excel('/content/flood dataset.xlsx')
```

Activity 3: Univariate analysis

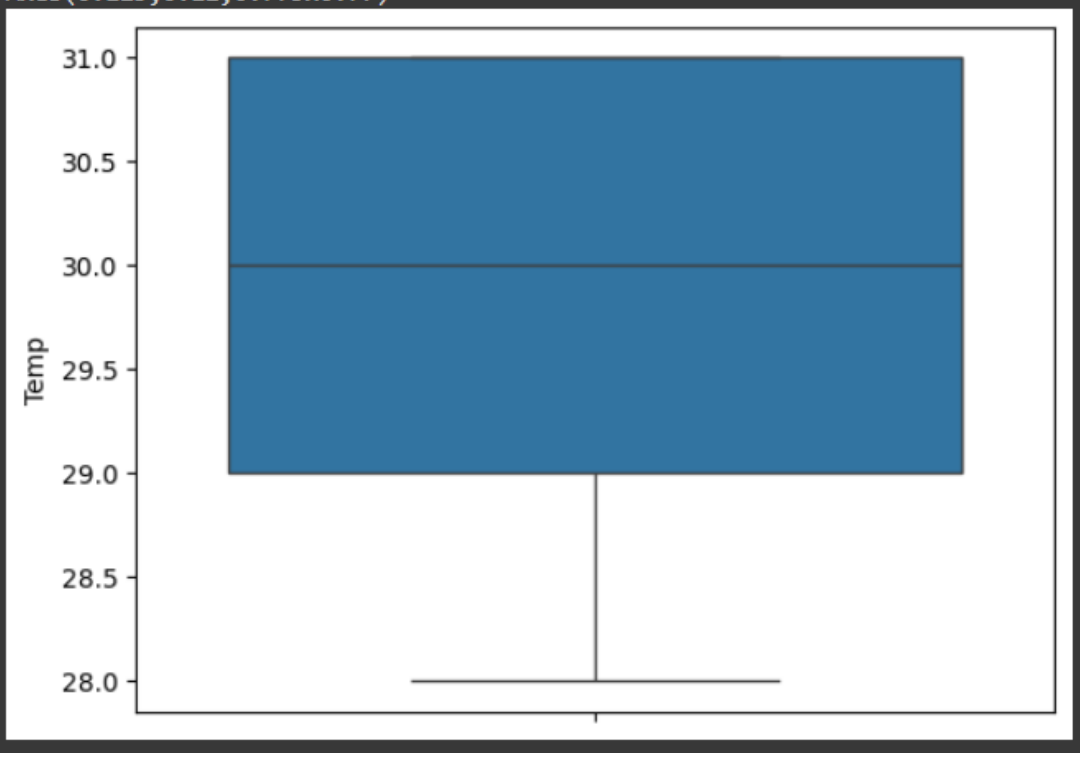
```
print(sns.distplot(data['Temp']))
```

```
<ipython-input-6-85f60c60e185>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  print(sns.distplot(data['Temp']))
Axes(0.125,0.11;0.775x0.77)
```



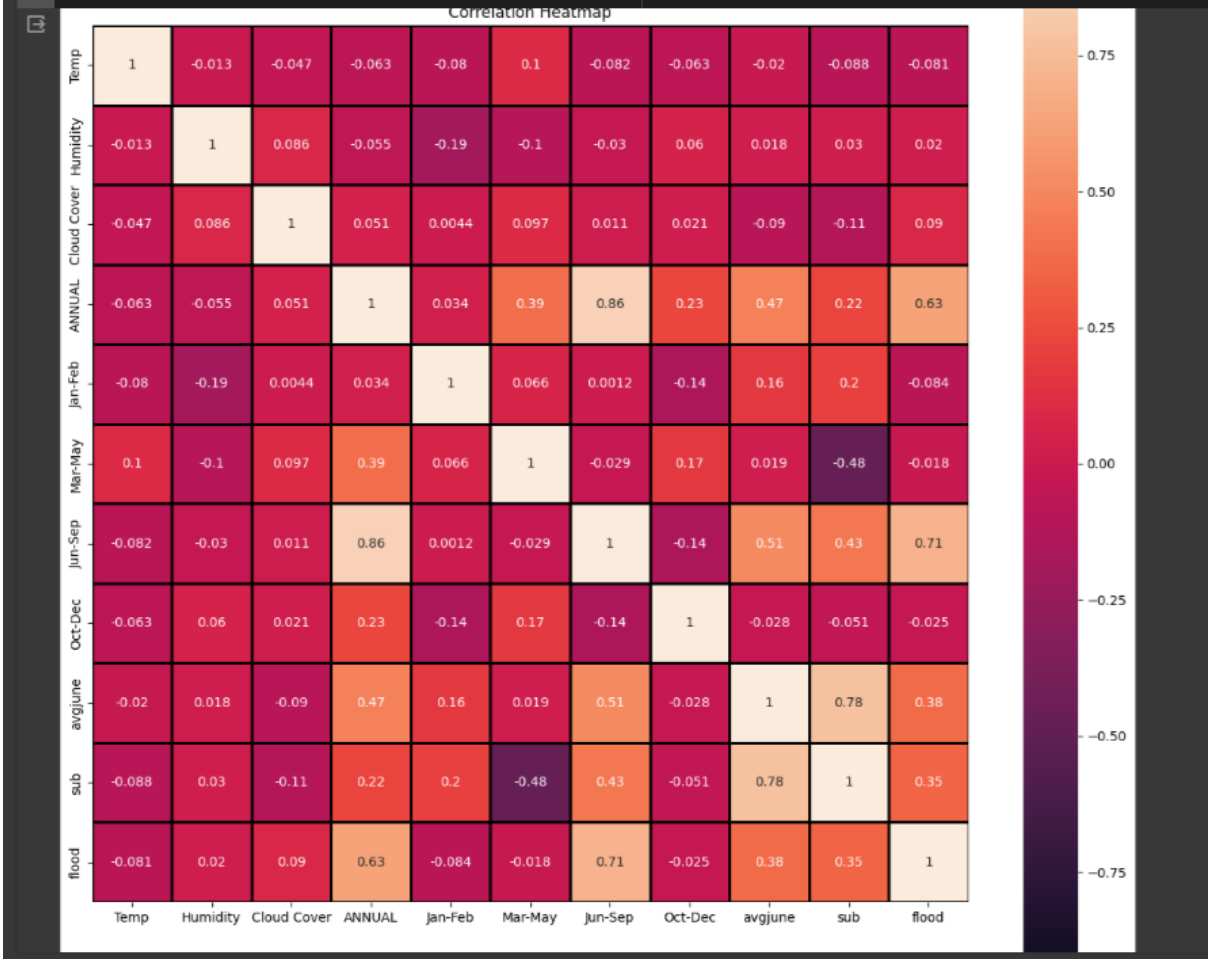
```
[7]  print(sns.boxplot(data['Temp']))
```

```
Axes(0.125,0.11;0.775x0.77)
```



Activity 4: Multivariate analysis

```
fig, ax = plt.subplots(figsize=(15, 15))

# Use sns.heatmap to create the heatmap
heatmap = sns.heatmap(data.corr(), annot=True, linewidths=1, linecolor='k', square=True,
                      vmin=-1, vmax=1, cbar_kws={"orientation": "vertical"}, cbar=True)

# Setting title
plt.title("Correlation Heatmap")

# Display the plot
plt.show()
```



Activity 5: Descriptive analysis

```
[10] data.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

```
[11] data.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
```

```
[12] data.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

Activity 1: Handling Missing values

```
[13] data.isnull().sum()
```

```
Temp           0
Humidity       0
Cloud Cover    0
ANNUAL         0
Jan-Feb        0
Mar-May        0
Jun-Sep        0
Oct-Dec        0
avgjune        0
sub            0
flood          0
dtype: int64
```

Activity 2: Handling outliers: Not required

Activity 3: Handling Categorical Values: all are numerical value

Activity 4: Splitting the Dataset into Dependent and Independent variables.

```
x=data.iloc[:,2:7].values #independent variable
y=data.iloc[:9:].values #dependent variable
```

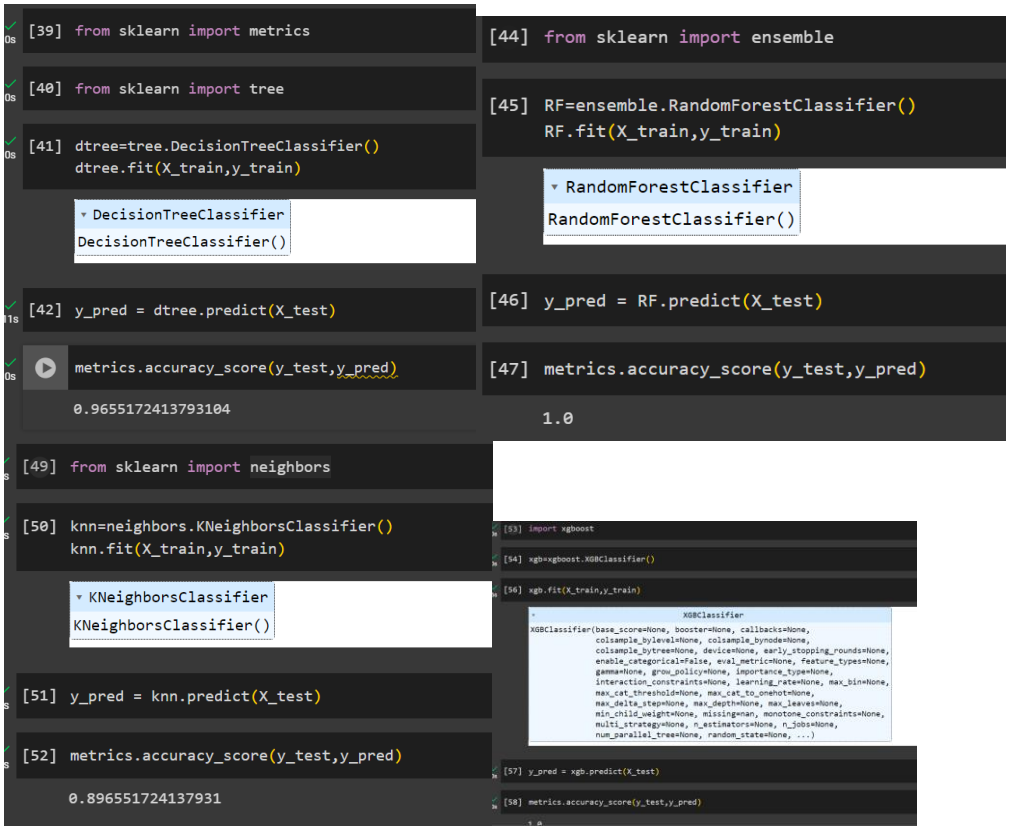
Activity 5: Split the dataset into Train set and Test set

```
[14] 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)

[15] X_train.shape

     (86, 5)

[16] X_test.shape

     (29, 5)

[17] y_train.shape

     (86, 2)

     y_test.shape

     (29, 2)
```

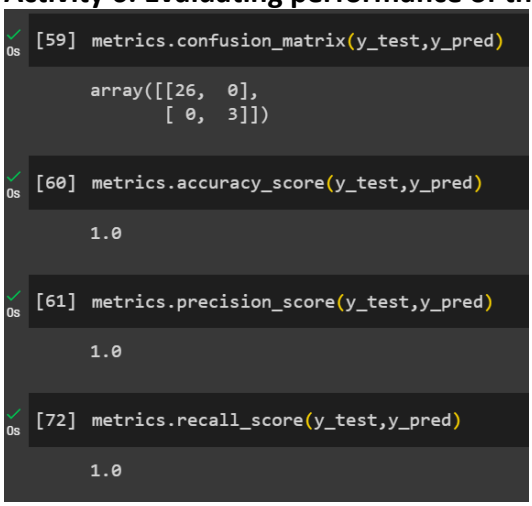
Activity 6: Feature scaling

```
[22] from sklearn.preprocessing import StandardScaler
     sc=StandardScaler()
     X_train=sc.fit_transform(X_train)
     X_test=sc.fit_transform(X_test)

[24] from joblib import dump
     dump(sc,"transform.save")

     ['transform.save']
```

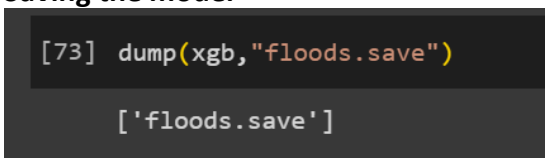
Milestone 4: Model Building



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

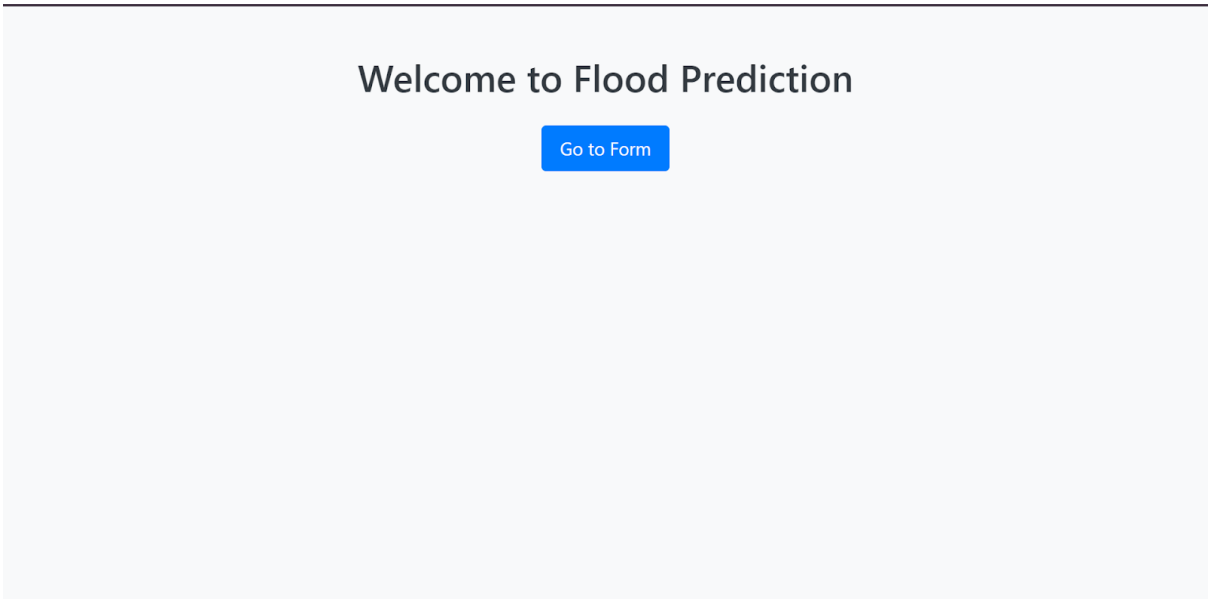


Saving the model

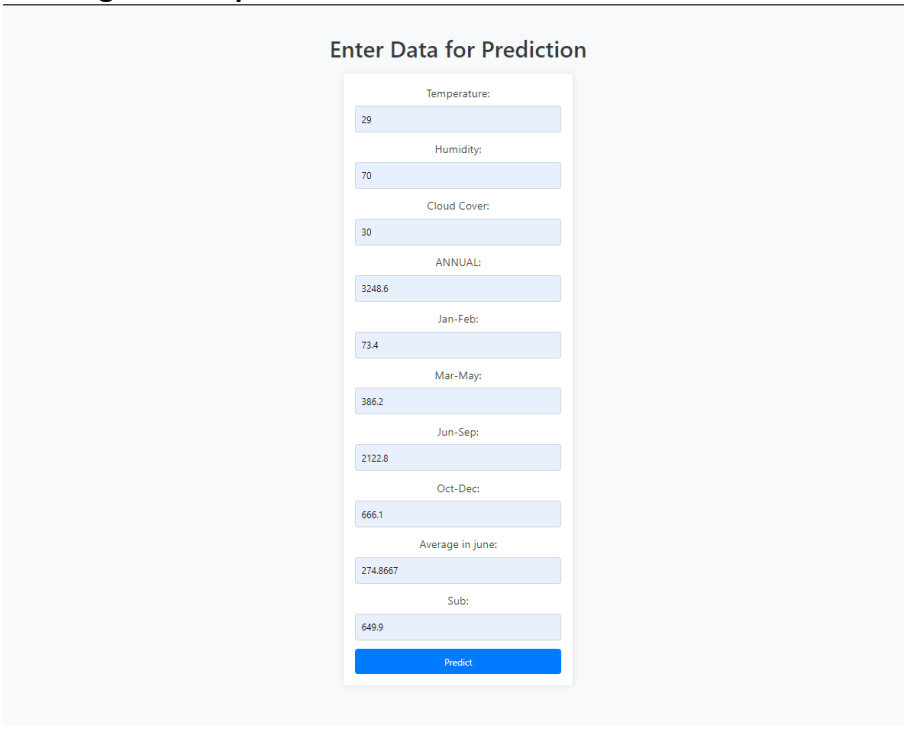


Milestone-5: Build Flask Application

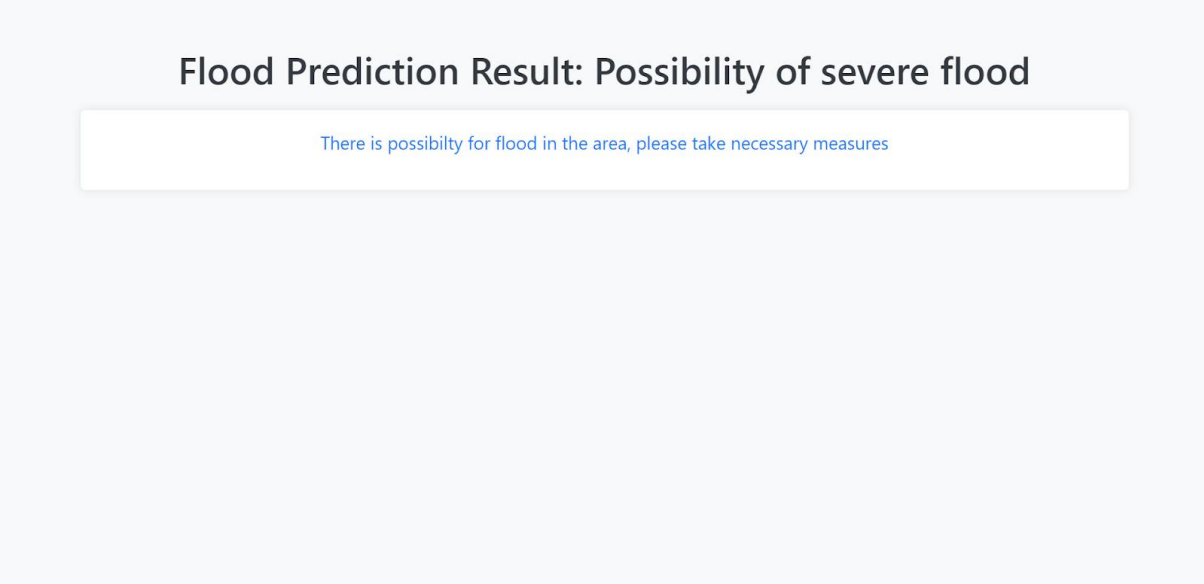
Home page:



Entering data for prediction:



Predicting flood:



Predicting no flood: