Rising Waters: A Machine Learning Approach to Flood Prediction

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS89043
PROJECT NAME	Rising Waters: A Machine Learning Approach to Flood Prediction
MAXIMUM MARKS	4 MARKS

6.4 MODEL BUILDING:

Decision tree model

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

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

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

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

How Decision Tree Works

- The model splits the dataset into branches based on feature values.
- It selects the best feature to split the data.
- The splitting continues until it reaches a final decision (leaf node).
- Final output is a predicted class (Flood or No Flood).

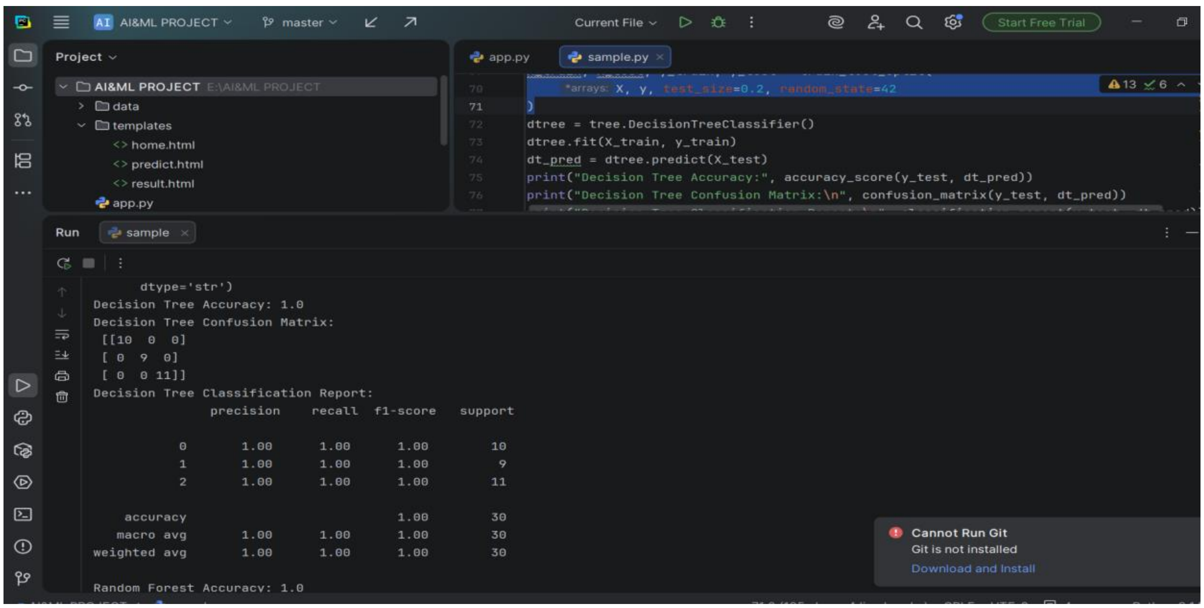
Training Process

- The training data (X_train, y_train) is given to the model.

- The model learns patterns and relationships between features and target.
- It creates rules to classify new data.

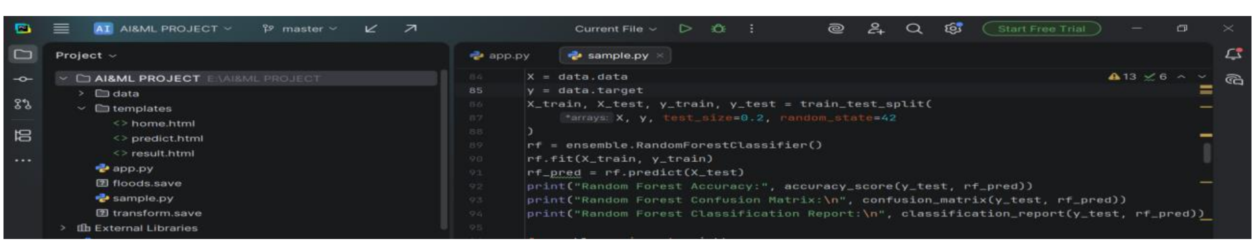
Testing Process

- Test data (X_test) is given.
- The trained model predicts output.
- Predictions are compared with actual values (y_test).



Random forest model

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



Rising Waters: A Machine Learning Approach to Flood Prediction

What is Random Forest

Random Forest is an ensemble learning method.
It builds multiple Decision Trees and combines their results to make a final prediction.

Instead of relying on a single tree:

- Many trees are created.
- Each tree gives a prediction.
- The final output is decided by majority voting (for classification).

Function Creation

A function named randomForest is created.
The training and testing datasets are passed as parameters:

- X_train
- X_test
- y_train
- y_test

Training Process

- The RandomForestClassifier algorithm is initialized.
- The training data is passed to the model using the .fit() function.
- The model builds multiple decision trees using different subsets of data and features.
- It learns patterns from rainfall, temperature, humidity, etc.

Testing Process

- The test data is passed to the trained model.
- The .predict() function is used to generate predictions.
- The predicted values are stored in a new variable.

Model Evaluation

Confusion Matrix

Used to measure:

- True Positive (Correct flood prediction)
- True Negative (Correct no flood prediction)
- False Positive (Incorrect flood prediction)
- False Negative (Missed flood case)

It helps understand how well the model is performing.

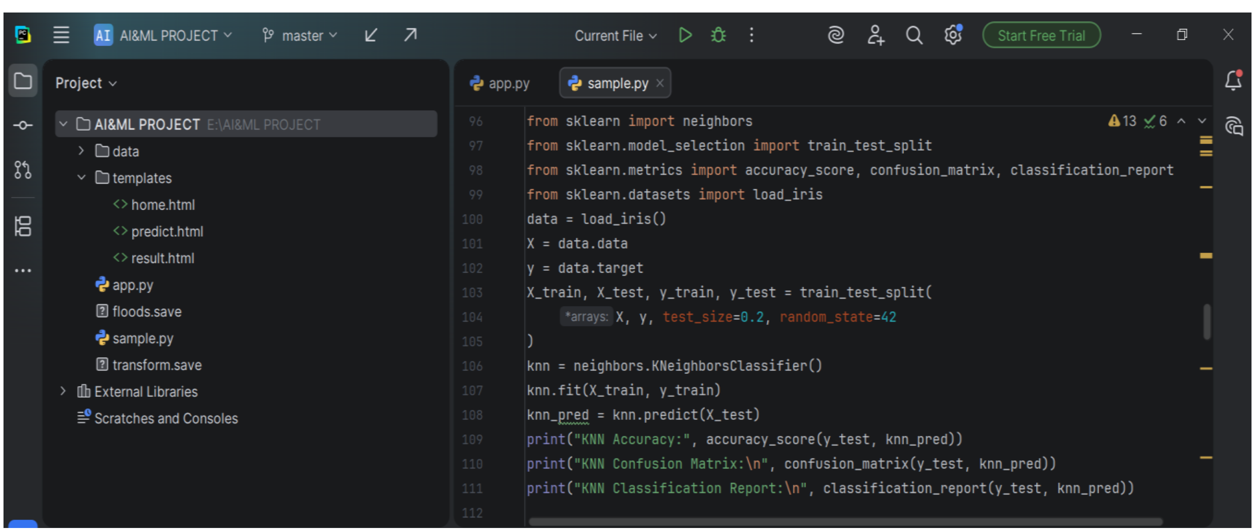
Classification Report

Provides performance metrics such as:

- **Accuracy** – Overall correctness of predictions
- **Precision** – Correctness of predicted flood cases
- **Recall** – Ability to detect actual flood cases
- **F1-score –** Balance between precision and recall

KNN model

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



KNN is a distance-based algorithm.
It classifies a new data point based on the K closest data points (neighbors) in the training dataset.

It works on a simple idea:

- Find the nearest K data points.
- Check their class labels.
- Assign the majority class to the new data point.

Function Creation

A function named KNN is created.
The training and testing datasets are passed as parameters:

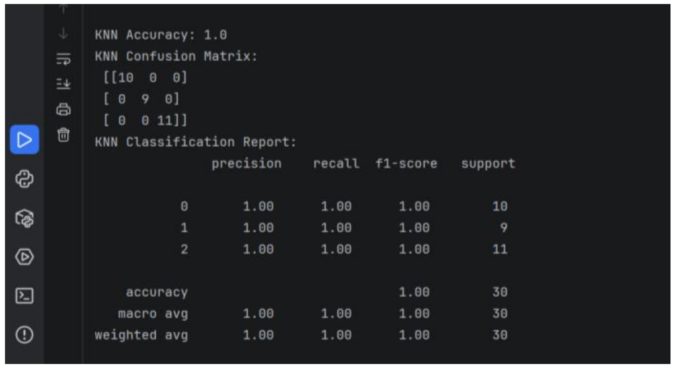
- X_train
- X_test
- y_train
- y_test

Training Process

- The KNeighborsClassifier algorithm is initialized.
- The training data is passed using the .fit() function.
- Unlike Decision Tree or Random Forest, KNN does not build a model.
- It simply stores the training data for future comparisons.

Testing Process

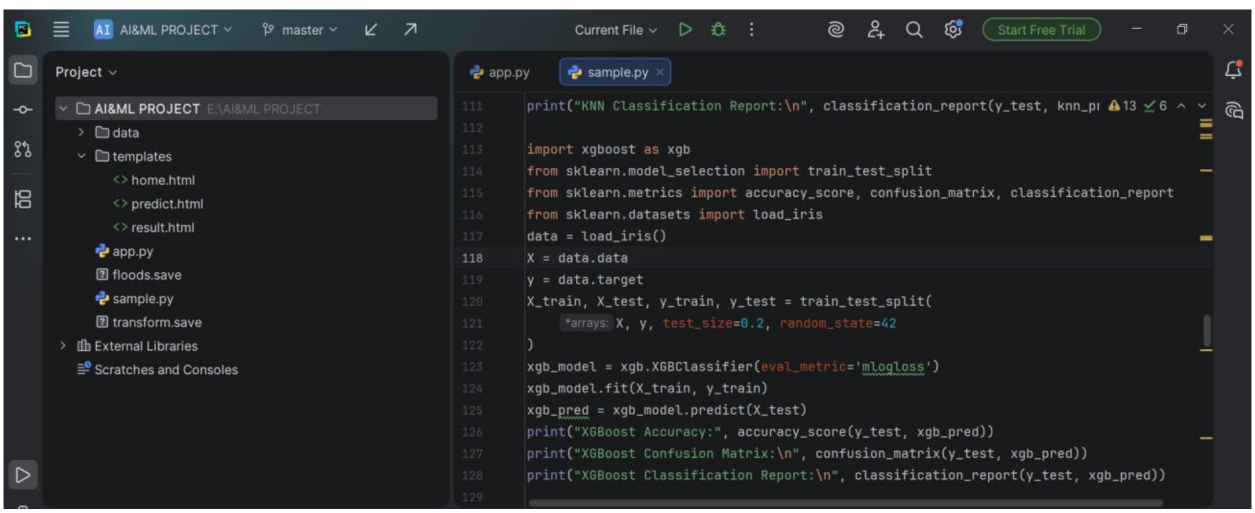
- The test data is passed to the model.
- The .predict() function calculates distances between test points and training points.
- It selects the K nearest neighbors.
- The majority class among neighbors is assigned as the prediction.
- The predicted values are stored in a new variable.



Xgboost model

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

Fit the model using x_train, y_train data



Boosting is an **ensemble learning technique** where:

- Multiple weak models (usually small decision trees) are built.
- Each new model tries to correct the errors made by the previous model.
- All models are combined to produce a strong final prediction.

Unlike Random Forest (which builds trees independently), boosting builds trees **sequentially**.

Model Initialization

Inside the function:

- The GradientBoostingClassifier algorithm is initialized.
- It creates multiple weak decision trees.
- Each tree improves the performance of the previous one.

Model Fitting (Training Phase)

The model is trained using:

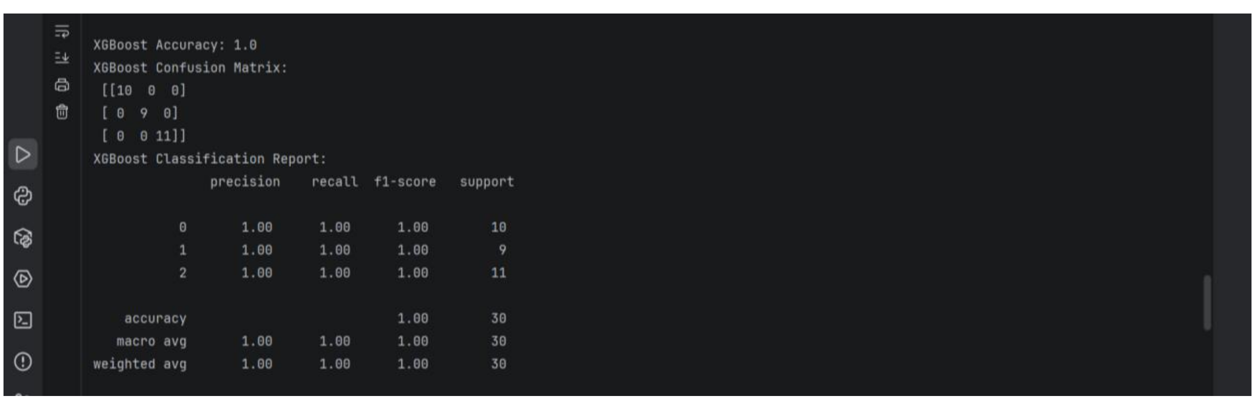
- x_train (input features)
- y_train (target variable)

Using the .fit(x_train, y_train) function:

- The model learns patterns from rainfall, humidity, temperature, etc.
- It minimizes prediction errors step by step.
- Each new tree focuses more on incorrectly predicted flood cases.

Testing Phase

- The test dataset (x_test) is given to the trained model.
- Predictions are generated using .predict(x_test).
- The predicted values are stored in a new variable.



```
XGBoost Accuracy: 1.0
XGBoost Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
XGBoost Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Compare the model

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

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

```
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
```

OUTPUT:

```
Run  sample

    accuracy                          1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Model 1 Accuracy: 1.0
Model 2 Accuracy: 1.0
Model 3 Accuracy: 1.0
Model 4 Accuracy: 1.0
[[ 7  0]
 [ 1 21]]
0.9655172413793104
1.0
0.9545454545454546
```

Evaluating performance of the model

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

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

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

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

         0.9655172413793104

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

         0.75

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

         1.0
```

OUTPUT:

```
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
```

Saving the model

- Joblib of save is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.
- Save our model by importing joblib dump class.

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

['floods.save']
```

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

FULL CODE OF MODEL BUILDING:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sea


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

print(dataset)


import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

sns.histplot(dataset['Temp'], kde=True)

plt.show()


sns.boxplot(x=dataset['Temp'])

plt.show()


import seaborn as sns

import matplotlib.pyplot as plt

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
)
plt.show()


dataset.head()


print(dataset.info())


print(dataset.describe().T)


print(dataset.isnull().any())
```

```
X = dataset.iloc[:, 2:7].values

y = dataset.iloc[:, 9].values


print(dataset.columns)


from sklearn.model_selection import train_test_split

X_train, X_test = train_test_split(dataset, test_size=0.25,
random_state=10)


from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


from joblib import dump

dump(sc, "transform.save")


from sklearn import tree

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data
```

```
y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

dtree = tree.DecisionTreeClassifier()

dtree.fit(X_train, y_train)

dt_pred = dtree.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, dt_pred))

print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test,
dt_pred))

print("Decision Tree Classification Report:\n",
classification_report(y_test, dt_pred))



from sklearn import ensemble

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)
```

```
rf = ensemble.RandomForestClassifier()

rf.fit(X_train, y_train)

rf_pred = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))

print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test,
rf_pred))

print("Random Forest Classification Report:\n",
classification_report(y_test, rf_pred))



from sklearn import neighbors

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

knn = neighbors.KNeighborsClassifier()

knn.fit(X_train, y_train)

knn_pred = knn.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, knn_pred))
```

```
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))

print("KNN Classification Report:\n", classification_report(y_test,
knn_pred))



import xgboost as xgb

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

xgb_model = xgb.XGBClassifier(eval_metric='mlogloss')

xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)

print("XGBoost Accuracy:", accuracy_score(y_test, xgb_pred))

print("XGBoost Confusion Matrix:\n", confusion_matrix(y_test,
xgb_pred))

print("XGBoost Classification Report:\n",
classification_report(y_test, xgb_pred))



# Import libraries
```

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn import metrics

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)

model1 = LogisticRegression(max_iter=200)

model1.fit(X_train, y_train)

p1 = model1.predict(X_test)

model2 = DecisionTreeClassifier()

model2.fit(X_train, y_train)

p2 = model2.predict(X_test)

model3 = RandomForestClassifier()

model3.fit(X_train, y_train)

p3 = model3.predict(X_test)
```

```
model4 = SVC()

model4.fit(X_train, y_train)

p4 = model4.predict(X_test)

print("Model 1 Accuracy:", metrics.accuracy_score(y_test, p1))

print("Model 2 Accuracy:", metrics.accuracy_score(y_test, p2))

print("Model 3 Accuracy:", metrics.accuracy_score(y_test, p3))

print("Model 4 Accuracy:", metrics.accuracy_score(y_test, p4))


from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn import metrics

data = load_breast_cancer()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=29, random_state=2

)

model = LogisticRegression(max_iter=5000)

model.fit(X_train, y_train)

p4 = model.predict(X_test)

print(metrics.confusion_matrix(y_test, p4))

print(metrics.accuracy_score(y_test, p4))
```

```
print(metrics.precision_score(y_test, p4))

print(metrics.recall_score(y_test, p4))


import pandas as pd

import joblib

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

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

if df["flood"].dtype == object:

    df["flood"] = df["flood"].map({"Yes": 1, "No": 0})

print("Flood value counts:")

print(df["flood"].value_counts())

X = df[[

    "Cloud Cover",

    "ANNUAL",

    "Jan-Feb",

    "Mar-May",

    "Jun-Sep"

]]

y = df["flood"]

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42
```

```
)
model = RandomForestClassifier(
    n_estimators=500,
    max_depth=15,
    class_weight='balanced',
    random_state=42
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
joblib.dump(model, "floods.save")
print("\nModel retrained successfully!")
```