* Flood prediction and response system-
* Train\_model.ipynb-

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, GridSearchCV, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, auc, precision\_recall\_curve

from sklearn.linear\_model import LogisticRegression

from xgboost import XGBClassifier

import joblib

# Load the dataset

dataset\_path = "flood\_prediction\_dataset.csv"

data = pd.read\_csv(dataset\_path)

# Convert date to datetime

data['date'] = pd.to\_datetime(data['date'])

# Preprocess data

def preprocess\_data(data):

# Separate features and target

X = data.drop(['flood\_occurred', 'flood\_probability', 'date'], axis=1)

# Remove unwanted features

X = X[['rainfall\_24h', 'rainfall\_72h', 'river\_level', 'soil\_moisture', 'reservoir\_level', 'previous\_floods', 'temperature']]

y = data['flood\_occurred']

# Create preprocessing steps

numerical\_transformer = StandardScaler()

# Create column transformer

preprocessor = ColumnTransformer(

transformers=[('num', numerical\_transformer, X.columns)])

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

return X\_train, X\_test, y\_train, y\_test, preprocessor, X.columns

# Main execution

def main():

print("\nPreprocessing data...")

X\_train, X\_test, y\_train, y\_test, preprocessor, feature\_cols = preprocess\_data(data)

# Build and evaluate models

models = {

'Random Forest': Pipeline([

('preprocessor', preprocessor),

('classifier', RandomForestClassifier(random\_state=42))

]),

'Gradient Boosting': Pipeline([

('preprocessor', preprocessor),

('classifier', GradientBoostingClassifier(random\_state=42))

]),

'XGBoost': Pipeline([

('preprocessor', preprocessor),

('classifier', XGBClassifier(random\_state=42, use\_label\_encoder=False, eval\_metric='logloss'))

]),

'Logistic Regression': Pipeline([

('preprocessor', preprocessor),

('classifier', LogisticRegression(random\_state=42, max\_iter=1000))

])

}

best\_model = None

best\_roc\_auc = 0

for name, model in models.items():

print(f"\nTraining {name}...")

model.fit(X\_train, y\_train)

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_proba)

roc\_auc = auc(fpr, tpr)

print(f"{name} ROC-AUC: {roc\_auc:.4f}")

if roc\_auc > best\_roc\_auc:

best\_roc\_auc = roc\_auc

best\_model = model

print("\nSaving best model...")

joblib.dump(best\_model, "flood\_prediction\_model.pkl")

print("Model saved as 'flood\_prediction\_model.pkl'")

if \_\_name\_\_ == "\_\_main\_\_":

main()