**Crop yield prediction using machine learning model**

By Swastik Das,

B.Tech in Electrical Engineering,

3rd year, University of Calcutta

under the guidance of

**Ankit Lodh**

Associate Research Engineer,

IDEAS TIH, ISI Kolkata

Period of Internship: 14th Jan 2025 - 30th April 2025

Report submitted to: IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

**Abstract**

Crop yield prediction is a critical aspect of agricultural planning and food security, particularly in a diverse and agrarian country like India. This study proposes a machine learning model to predict yields of major crops like wheat, rice, mustard, gram, massor, and potato. The model uses historical yield data along with factors such as temperature, rainfall, water level, and live storage. Supervised learning algorithms are applied to understand complex relationships between crop yield and environmental variables. Region-wise and seasonal data are used to account for India's climatic diversity. The model shows strong predictive accuracy and can support better planning for farmers and policymakers. This data-driven approach promotes efficient resource use and contributes to sustainable agriculture.

**Introduction**

India’s agriculture plays a vital role in its economy, supporting more than half of its population. However, crop production is highly vulnerable to climatic fluctuations, irregular rainfall, and water availability from reservoirs. To address this challenge, this project leverages machine learning techniques to predict the annual yield of major Indian crops such as wheat, rice, mustard, gram, masoor, and potato based on historical data and agro-environmental variables.

**Purpose**

The objective of the project is to build predictive models that estimate crop yield for 2023 using key factors like:

* Previous years’ yields (yield data upto 2022)
* Monthly and yearly temperature patterns
* Rainfall data
* Water reservoir indicators such as water level and live storage

**Technologies Used**

* Programming: Python
* Platform: Google Colaboratory
* Libraries: Pandas, NumPy, scikit-learn, matplotlib
* Models: Random Forest Regressor, XGB Regressor

**Methodology**

1. **Data Preparation**: Cleaned and aggregated historical weather, reservoir, and yield data on a monthly and yearly basis.
2. **Feature Engineering**: Created lag features, rolling averages, and interaction terms to capture temporal trends and dependencies.
3. **Model Training**: Trained separate models for each state using Random Forest and XGBoost on data up to 2022.
4. **Forecasting**: Predicted monthly environmental factors for 2023 are then used to predict annual yield for 2023.

**Project Objective**

The main objectives of this project are :

* To develop machine learning models (Random Forest and XGBoost) that can accurately predict annual crop yields in Indian states using historical yield and environmental data (like temperature, rainfall, reservoir).
* To analyse the relationship between crop yield and key influencing factors such as temperature, rainfall, water level, and live storage, and to determine how these variables interact over time.
* To illustrate the feasibility of forecasting crop yields using predicted environmental data for future years, enabling proactive agricultural planning even without future yield data.
* To build a scalable, per-state prediction framework that accounts for regional variations in climate, water availability, and crop response across India.

**Methodology**

**1.Machine Learning Model Using Random Forest**

1. Data Loading and Preparation - Imports necessary libraries and loads the dataset (in this case, wheat data).
2. Feature Engineering - Converts string dates to datetime objects and creates separate columns for year, month, and day.
3. Feature Selection and Cleaning - Selects relevant features and cleans the data.
4. Advanced Feature Engineering - Creates time-series features to improve model performance like lag and rolling mean.
5. Model Training and Evaluation - Trains and evaluates state-specific models using Random Forest regression.
6. Results Presentation - Organizes and displays the comparison results.
7. State Encoding - Converts categorical state names to numerical codes.
8. Feature Prediction for 2023 - Predicts weather/reservoir features for 2023 by using time and state code to predict each feature.
9. Feature Generation for year 2023 - Generates monthly predictions for all states in 2023.
10. Yield Prediction for 2023 - Predicts crop yields for 2023 using the trained models by using predicted 2023 features as input.
11. Data preparation for Visualization - Combines historical and predicted data for visualization of yield trends.

**2.Machine Learning Model Using XGBoost**

1. Load and Preprocess Monthly Data - Loads and prepares the dataset for analysis by converting date strings to datetime objects and extracts year/month and handles missing values in reservoir data using forward then backward filling within each state.
2. Train Monthly Models to Forecast 2023 Variables - Forecasts monthly weather and reservoir variables for 2023 by creating separate models for each target variable (temperature, rainfall, etc.)
3. Aggregate Forecasted Monthly Data to Annual(2023) - Converts monthly forecasts to annual aggregates.
4. Combine Historical Data with Forecasted(2023) Data - Creates a unified dataset of historical and forecasted data.
5. Feature Engineering : Lag, Rolling, Interactions - Creates lag features (1, 2, and 3 year lags) for key variables, adds rolling 3-year averages, creates interaction feature between temperature and rainfall, includes raw year as a feature to capture long-term trends and all features are calculated within each state's time series.
6. Predict Yield(2022 + 2023 forecast) - Builds yield prediction models and generates forecasts.
7. Display Results - Shows actual vs predicted yields for 2022 with RMSE and also displays clean table of 2023 yield forecasts.
8. Plot of Actual vs Predicted Yields(2000-2023) - Visualizes yield trends and forecasts.

**Data Analysis and Results**

The two Supervised Machine Learning models – Random Forest Regressor and XGBoost Regressor has successfully predicted yields of all major crops grown in India in the year 2022 and 2023. The RMSE obtained per state for year 2022 helps us to analyse the models’ performance based on each state.

A detailed report with the yield predictions and the RMSE per state is given by 🡺 [Click Here](file:///C:\Users\swast\Downloads\IDEAS%20ISI%20FASAL%20ML\Swastik_das_project_analysis.docx)

**Conclusion**

This study demonstrates the effectiveness of machine learning models, specifically Random Forest and XGBoost, in predicting the yield of important Indian crops such as wheat, rice, mustard, gram, masoor, and potato. By leveraging historical yield data along with key agro-environmental factors like temperature, rainfall, water level, and live storage, the models achieved reliable and accurate yield forecasts. These predictions can significantly aid farmers, agricultural planners, and policymakers in making informed decisions related to crop planning, resource allocation, and risk management.

The results highlight the potential of data-driven approaches in addressing agricultural challenges in a climate-sensitive country like India. For future work, the model can be improved by incorporating additional real-time data such as soil health, pest incidence, satellite imagery, and economic factors. Moreover, developing region-specific models and integrating deep learning techniques could further enhance prediction accuracy. This research lays the foundation for smarter, technology-driven agriculture in India.

**Random Forest CODE**

# === Part 1: Actual vs Predicted Yields for 2022 ===

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

file1="/content/merged\_wheat\_reservoir.csv"

file2="/content/merged\_gram\_reservoir.csv"

file3="/content/merged\_massor\_reservoir.csv"

file4="/content/merged\_potato\_reservoir.csv"

file5="/content/merged\_rabi\_rice\_reservoir.csv"

file6="/content/merged\_mustard\_reservoir.csv"

df = pd.read\_csv(file1)

df['temperature\_recorded\_date'] = pd.to\_datetime(df['temperature\_recorded\_date'])

df['year'] = df['temperature\_recorded\_date'].dt.year

df['month'] = df['temperature\_recorded\_date'].dt.month

df['day'] = df['temperature\_recorded\_date'].dt.day

features\_base = [

    'state\_temperature\_max\_val',

    'state\_temperature\_min\_val',

    'state\_rainfall\_val',

    'Level',

    'Current Live Storage'

]

target = 'yield'

df\_clean = df.dropna(subset=features\_base + [target, 'state\_name'])

df\_clean = df\_clean.sort\_values(by='temperature\_recorded\_date')

df\_clean['lag1'] = df\_clean.groupby('state\_name')['yield'].shift(1)

df\_clean['rolling\_mean\_7'] = df\_clean.groupby('state\_name')['yield'].transform(

    lambda x: x.shift(1).rolling(window=7, min\_periods=1).mean()

)

df\_final = df\_clean.dropna(subset=['lag1', 'rolling\_mean\_7'])

features = features\_base + ['month', 'day', 'lag1', 'rolling\_mean\_7']

min\_records\_threshold = 50

eligible\_states = df\_final['state\_name'].value\_counts()

eligible\_states = eligible\_states[eligible\_states >= min\_records\_threshold].index.tolist()

state\_rmse = {}

yield\_comparison = {}

for state in eligible\_states:

    state\_data = df\_final[df\_final['state\_name'] == state]

    train\_data = state\_data[state\_data['year'] < 2022]

    test\_data = state\_data[state\_data['year'] == 2022]

    if train\_data.empty or test\_data.empty:

        continue

    X\_train = train\_data[features]

    y\_train = train\_data[target]

    X\_test = test\_data[features]

    y\_test = test\_data[target]

    model = RandomForestRegressor(n\_estimators=100, random\_state=42)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

    state\_rmse[state] = rmse

    comparison\_df = test\_data[['year']].copy()

    comparison\_df['Actual Yield'] = y\_test.values

    comparison\_df['Predicted Yield'] = y\_pred

    comparison\_df = comparison\_df.groupby('year').mean().reset\_index()

    yield\_comparison[state] = comparison\_df

results = []

for state in sorted(yield\_comparison.keys()):

    row = yield\_comparison[state]

    year = int(row['year'].values[0])

    actual\_yield = row['Actual Yield'].values[0]

    predicted\_yield = row['Predicted Yield'].values[0]

    rmse = state\_rmse[state]

    results.append({

        'state': state,

        'year': year,

        'actual\_yield': actual\_yield,

        'predicted\_yield': predicted\_yield,

        'rmse': rmse

    })

results\_df = pd.DataFrame(results)

results\_df.set\_index(['state', 'year'], inplace=True)

results\_df.sort\_index(inplace=True)

print("\nPredicted vs Actual Yields (2022):\n")

print(results\_df)

print("\nAverage RMSE per State:\n")

print(results\_df['rmse'].sort\_values())

# === Part 2: Predict Yield for 2023 ===

from sklearn.preprocessing import LabelEncoder

df = pd.read\_csv(file1)

df['temperature\_recorded\_date'] = pd.to\_datetime(df['temperature\_recorded\_date'])

df['year'] = df['temperature\_recorded\_date'].dt.year

df['month'] = df['temperature\_recorded\_date'].dt.month

df['day'] = df['temperature\_recorded\_date'].dt.day

le = LabelEncoder()

df['state\_code'] = le.fit\_transform(df['state\_name'])

df = df.dropna(subset=features\_base + [target, 'state\_name'])

df = df.sort\_values(by=['state\_name', 'temperature\_recorded\_date'])

df['lag1'] = df.groupby('state\_name')[target].shift(1)

df['rolling\_mean\_7'] = df.groupby('state\_name')[target].transform(

    lambda x: x.shift(1).rolling(window=7, min\_periods=1).mean()

)

df\_final = df.dropna(subset=['lag1', 'rolling\_mean\_7'])

eligible\_states = df\_final['state\_name'].value\_counts()

eligible\_states = eligible\_states[eligible\_states >= min\_records\_threshold].index.tolist()

features\_for\_monthly = ['year', 'month', 'state\_code']

predicted\_2023 = []

for target\_col in features\_base:

    model = RandomForestRegressor(n\_estimators=100, random\_state=42)

    df\_train = df[df['year'] <= 2022].dropna(subset=[target\_col])

    X = df\_train[features\_for\_monthly]

    y = df\_train[target\_col]

    model.fit(X, y)

    state\_codes = df['state\_code'].unique()

    months = list(range(1, 13))

    pred\_rows = pd.DataFrame([

        {'year': 2023, 'month': m, 'state\_code': s}

        for s in state\_codes for m in months

    ])

    preds = model.predict(pred\_rows)

    pred\_rows[target\_col] = preds

    predicted\_2023.append(pred\_rows)

df\_2023\_features = predicted\_2023[0]

for df\_feat in predicted\_2023[1:]:

    df\_2023\_features = df\_2023\_features.merge(df\_feat, on=['year', 'month', 'state\_code'])

df\_2023\_features['state\_name'] = le.inverse\_transform(df\_2023\_features['state\_code'])

df\_2023\_state\_avg = df\_2023\_features.groupby('state\_name')[features\_base].mean().reset\_index()

df\_2023\_state\_avg['year'] = 2023

df\_2023\_state\_avg['month'] = 6

df\_2023\_state\_avg['day'] = 15

yield\_predictions = []

for state in eligible\_states:

    state\_data = df\_final[df\_final['state\_name'] == state]

    train\_data = state\_data[state\_data['year'] <= 2022]

    if train\_data.empty:

        continue

    model = RandomForestRegressor(n\_estimators=100, random\_state=42)

    X\_train = train\_data[features]

    y\_train = train\_data[target]

    model.fit(X\_train, y\_train)

    test\_row = df\_2023\_state\_avg[df\_2023\_state\_avg['state\_name'] == state].copy()

    if test\_row.empty: continue

    train\_sorted = train\_data.sort\_values(by='temperature\_recorded\_date')

    last\_3\_years\_data = train\_sorted[train\_sorted['year'].isin([2020, 2021, 2022])]

    lag1\_value = last\_3\_years\_data['yield'].mean()

    rolling\_mean\_value = (

        train\_sorted.groupby('state\_name')['yield']

        .rolling(window=7, min\_periods=1).mean()

        .reset\_index(level=0, drop=True)

        .iloc[-50:]

        .mean()

    )

    test\_row['lag1'] = lag1\_value

    test\_row['rolling\_mean\_7'] = rolling\_mean\_value

    X\_test = test\_row[features]

    predicted\_yield = model.predict(X\_test)[0]

    yield\_predictions.append({

        'state\_name': state,

        'predicted\_yield\_2023': predicted\_yield

    })

df\_yield\_2023 = pd.DataFrame(yield\_predictions).sort\_values(by='predicted\_yield\_2023', ascending=False)

print("\nPredicted Yield for 2023:\n")

print(df\_yield\_2023.to\_string(index=False))

# === Part 3: Plot All States (2000–2023) ===

# Combine past actuals + 2023 prediction

plot\_df = df\_final[df\_final['state\_name'].isin(eligible\_states)]

plot\_df\_grouped = plot\_df.groupby(['state\_name', 'year'])['yield'].mean().reset\_index()

pred\_2023\_df = df\_yield\_2023.rename(columns={'state\_name': 'state\_name', 'predicted\_yield\_2023': 'yield'})

pred\_2023\_df['year'] = 2023

combined\_df = pd.concat([plot\_df\_grouped, pred\_2023\_df[['state\_name', 'year', 'yield']]], ignore\_index=True)

states = sorted(combined\_df['state\_name'].unique())

n\_states = len(states)

n\_cols = 4

n\_rows = (n\_states + n\_cols - 1) // n\_cols

fig, axs = plt.subplots(n\_rows, n\_cols, figsize=(5 \* n\_cols, 4 \* n\_rows), constrained\_layout=True)

for i, state in enumerate(states):

    row, col = divmod(i, n\_cols)

    ax = axs[row][col] if n\_rows > 1 else axs[col]

    state\_data = combined\_df[combined\_df['state\_name'] == state]

    ax.plot(state\_data['year'], state\_data['yield'], marker='o', label='Yield')

    ax.axvline(2022, color='gray', linestyle='--', linewidth=1)

    ax.set\_title(state)

    ax.set\_xlabel("Year")

    ax.set\_ylabel("Yield")

    ax.grid(True)

plt.suptitle("Actual Yields (2000–2022) and Predicted 2023 Yield by State", fontsize=16)

plt.show()

**XGBoost CODE**

# ==============================

# 1. Load and Preprocess Monthly Data

# ==============================

import pandas as pd

import numpy as np

from xgboost import XGBRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

import warnings

warnings.filterwarnings("ignore")

# Load dataset

file1="/content/merged\_wheat\_reservoir.csv"

file2="/content/merged\_gram\_reservoir.csv"

file3="/content/merged\_massor\_reservoir.csv"

file4="/content/merged\_potato\_reservoir.csv"

file5="/content/merged\_rabi\_rice\_reservoir.csv"

file6="/content/merged\_mustard\_reservoir.csv"

df = pd.read\_csv(file1)

# Date conversion

df['temperature\_recorded\_date'] = pd.to\_datetime(df['temperature\_recorded\_date'])

df['year'] = df['temperature\_recorded\_date'].dt.year

df['month'] = df['temperature\_recorded\_date'].dt.month

# Fill missing values in exogenous variables

exog\_vars = ['Level', 'Current Live Storage']

df[exog\_vars] = df.groupby('state\_name')[exog\_vars].transform(lambda g: g.ffill().bfill())

# ==============================

# 2. Train Monthly Models to Forecast 2023 Variables

# ==============================

monthly\_targets = ['state\_temperature\_max\_val', 'state\_temperature\_min\_val',

                   'state\_rainfall\_val', 'Level', 'Current Live Storage']

forecasted\_rows = []

for state in df['state\_name'].unique():

    state\_df = df[df['state\_name'] == state].copy()

    for target in monthly\_targets:

        for month in range(1, 13):

            temp\_df = state\_df[state\_df['month'] == month]

            if temp\_df['year'].nunique() < 4:

                continue

            train\_df = temp\_df[temp\_df['year'] < 2023]

            test\_df = pd.DataFrame({

                'year': [2023],

                'month': [month],

                'state\_name': [state]

            })

            X\_train = train\_df[['year']]

            y\_train = train\_df[target]

            X\_test = test\_df[['year']]

            model = XGBRegressor(n\_estimators=100, max\_depth=3, learning\_rate=0.1, random\_state=42)

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

            pred = model.predict(X\_test)[0]

            forecasted\_rows.append({

                'state\_name': state,

                'year': 2023,

                'month': month,

                target: pred

            })

# Convert forecasted monthly data to DataFrame

forecast\_df = pd.DataFrame(forecasted\_rows)

# Pivot to reshape into one row per state-month

forecast\_df = forecast\_df.groupby(['state\_name', 'year', 'month']).first().reset\_index()

# ==============================

# 3. Aggregate Forecasted Monthly Data to Annual (2023)

# ==============================

yearly\_agg = forecast\_df.groupby(['state\_name', 'year']).agg({

    'state\_temperature\_max\_val': 'mean',

    'state\_temperature\_min\_val': 'mean',

    'state\_rainfall\_val': 'sum',

    'Level': 'mean',

    'Current Live Storage': 'mean'

}).reset\_index()

# Add dummy yield for 2023 (will not be used for training)

yearly\_agg['yield'] = np.nan

# ==============================

# 4. Combine Historical Annual Data with Forecasted 2023 Data

# ==============================

# Aggregate historical data

agg\_funcs = {

    'state\_temperature\_max\_val': 'mean',

    'state\_temperature\_min\_val': 'mean',

    'state\_rainfall\_val': 'sum',

    'yield': 'mean',

    'Level': 'mean',

    'Current Live Storage': 'mean'

}

annual\_df = df.groupby(['state\_name', 'year']).agg(agg\_funcs).reset\_index()

# Combine with forecasted 2023 data

full\_annual\_df = pd.concat([annual\_df, yearly\_agg], ignore\_index=True)

# ==============================

# 5. Feature Engineering: Lag, Rolling, Interactions

# ==============================

features\_to\_lag = ['state\_temperature\_max\_val', 'state\_temperature\_min\_val',

                   'state\_rainfall\_val', 'Level', 'Current Live Storage']

for feature in features\_to\_lag:

    full\_annual\_df[f'{feature}\_lag1'] = full\_annual\_df.groupby('state\_name')[feature].shift(1)

    full\_annual\_df[f'{feature}\_lag2'] = full\_annual\_df.groupby('state\_name')[feature].shift(2)

    full\_annual\_df[f'{feature}\_lag3'] = full\_annual\_df.groupby('state\_name')[feature].shift(3)

    full\_annual\_df[f'{feature}\_roll3'] = full\_annual\_df.groupby('state\_name')[feature].transform(lambda x: x.shift(1).rolling(3).mean())

# Interaction and trend feature

full\_annual\_df['temp\_rainfall\_interaction'] = (

    (full\_annual\_df['state\_temperature\_max\_val'] + full\_annual\_df['state\_temperature\_min\_val']) / 2

) \* full\_annual\_df['state\_rainfall\_val']

full\_annual\_df['year\_feature'] = full\_annual\_df['year']

# ==============================

# 6. Predict Yield for 2022 (with RMSE) and 2023 (Forecast)

# ==============================

model\_df = full\_annual\_df.dropna(subset=['yield'])

results = []

target = 'yield'

features = [col for col in model\_df.columns if col not in ['state\_name', 'year', 'yield']]

for state in model\_df['state\_name'].unique():

    state\_df = model\_df[model\_df['state\_name'] == state].copy()

    train\_df = state\_df[state\_df['year'] < 2022]

    test\_df\_2022 = state\_df[state\_df['year'] == 2022]

    test\_df\_2023 = full\_annual\_df[(full\_annual\_df['state\_name'] == state) & (full\_annual\_df['year'] == 2023)]

    if train\_df.empty: continue

    # Standardize

    scaler = StandardScaler()

    X\_train = scaler.fit\_transform(train\_df[features])

    y\_train = train\_df[target]

    model = XGBRegressor(n\_estimators=300, max\_depth=4, learning\_rate=0.05, random\_state=42)

    model.fit(X\_train, y\_train)

    # Predict 2022

    if not test\_df\_2022.empty:

        X\_test\_2022 = scaler.transform(test\_df\_2022[features])

        y\_test\_2022 = test\_df\_2022[target].values

        preds\_2022 = model.predict(X\_test\_2022)

        rmse\_2022 = np.sqrt(mean\_squared\_error(y\_test\_2022, preds\_2022))

        results.append({

            'state': state,

            'year': 2022,

            'actual\_yield': y\_test\_2022[0],

            'predicted\_yield': preds\_2022[0],

            'rmse': rmse\_2022

        })

    # Predict 2023

    if not test\_df\_2023.empty:

        X\_test\_2023 = scaler.transform(test\_df\_2023[features])

        pred\_2023 = model.predict(X\_test\_2023)[0]

        results.append({

            'state': state,

            'year': 2023,

            'actual\_yield': None,

            'predicted\_yield': pred\_2023,

            'rmse': None

        })

# ==============================

# 7. Display Results

# ==============================

results\_df = pd.DataFrame(results)

# Predicted vs Actual for 2022

pivot = results\_df[results\_df['year'] == 2022].pivot\_table(

    index=['state', 'year'],

    values=['actual\_yield', 'predicted\_yield', 'rmse'],

    aggfunc='first'

)

print("\n Predicted vs Actual Yields (2022):\n")

print(pivot.round(4))

# Average RMSE per State

rmse\_per\_state = results\_df[results\_df['year'] == 2022].groupby('state')['rmse'].mean().sort\_values()

print("\n Average RMSE per State:\n")

print(rmse\_per\_state.round(4))

# Forecasted 2023 Yields

preds\_2023\_df = results\_df[results\_df['year'] == 2023][['state', 'predicted\_yield']]

print("\n Predicted Yield for 2023:\n")

print(preds\_2023\_df.set\_index('state').round(4))