

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

```
faostat_df = pd.read_csv("/content/FAOSTAT_data_en_11-19-2025 .csv")
crop_df = pd.read_csv("/content/Crop_recommendation .csv")
```

```
faostat_india = faostat_df[faostat_df["Area"] == "India"].copy()
```

```
faostat_india["crop"] = (
    faostat_india["Item"]
    .str.split(",")
    .apply(lambda lst: [x.strip() for x in lst])
)
```

```
faostat_exploded = faostat_india.explode("crop").reset_index(drop=True)
```

```
faostat_exploded = faostat_exploded.rename(columns={"Item": "item"})
crop_df = crop_df.rename(columns={"label": "crop"})
```

```
print("Crop columns:", crop_df.columns)
print("FAOSTAT columns:", faostat_exploded.columns)
```

```
Crop columns: Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'crop'], dtype='object')
FAOSTAT columns: Index(['Domain Code', 'Domain', 'Area Code (M49)', 'Area', 'Element Code',
   'Element', 'Item Code (CPC)', 'item', 'Year Code', 'Year', 'Unit',
   'Value', 'Flag', 'Flag Description', 'Note', 'crop'],
   dtype='object')
```

```
label_to_fao = {
    "apple": "Apples",
    "banana": "Bananas",
    "chickpea": "Chick peas",
    "coconut": "Coconuts",
    "coffee": "Coffee",
    "cotton": "Seed cotton",
    "grapes": "Grapes",
    "jute": "Jute",
    "lentil": "Lentils",
    "maize": "Maize (corn)",
    "mango": "Mangoes",
    "mothbeans": "Beans",
    "muskmelon": "Cantaloupes and other melons",
    "orange": "Oranges",
    "papaya": "Papayas",
    "pigeonpeas": "Pigeon peas",
    "rice": "Rice",
    "watermelon": "Watermelons",
}
```

```
crop_df["FAO_name"] = crop_df["crop"].map(label_to_fao)
faostat_exploded["FAO_name"] = faostat_exploded["crop"]
```

```
crop_mapped = crop_df[~crop_df["FAO_name"].isna()].copy()
print("Mapped crops:", sorted(crop_mapped["crop"].unique()))
print("Number of mapped crops:", crop_mapped["crop"].nunique())
```

```
Mapped crops: ['apple', 'banana', 'chickpea', 'coconut', 'coffee', 'cotton', 'grapes', 'jute', 'lentil', 'maize', 'mango', 'moth'
Number of mapped crops: 18
```

```
merged = faostat_exploded.merge(
    crop_mapped,
    on="FAO_name",
```

```

how="inner",
suffixes=("_fao", "_ml")
)

merged["crop"] = merged["crop_ml"]

print("Merged shape:", merged.shape)
print("Unique crops after merge:", merged["crop"].nunique())
print(sorted(merged["crop"].unique()))

Merged shape: (129600, 26)
Unique crops after merge: 18
['apple', 'banana', 'chickpea', 'coconut', 'coffee', 'cotton', 'grapes', 'jute', 'lentil', 'maize', 'mango', 'mothbeans', 'muskm

```

```

merged_prod = merged[merged["Element"] == "Production"].copy()
print("Production shape:", merged_prod.shape)

```

Production shape: (43200, 26)

```

cols_keep = [
"Element", "Value", "N", "P", "K",
"temperature", "humidity", "ph", "rainfall", "crop"
]

```

```

final_df = merged_prod[cols_keep].copy()
print("Final DF shape:", final_df.shape)
print(final_df.head())

```

Final DF shape: (43200, 10)

	Element	Value	N	P	K	temperature	humidity	ph	\
200	Production	1050000.0	24	128	196	22.750888	90.694892	5.521467	
201	Production	1050000.0	7	144	197	23.849401	94.348150	6.133221	
202	Production	1050000.0	14	128	205	22.608010	94.589006	6.226290	
203	Production	1050000.0	8	120	201	21.186674	91.134357	6.321152	
204	Production	1050000.0	20	129	201	23.410447	91.699133	5.587906	

	rainfall	crop
200	110.431786	apple
201	114.051249	apple
202	116.039659	apple
203	122.233323	apple
204	116.077793	apple

```

print("Unique crops in final_df:")
print(sorted(final_df["crop"].unique()))
print("Number of unique crops:", final_df["crop"].nunique())

```

Unique crops in final\_df:  
['apple', 'banana', 'chickpea', 'coconut', 'coffee', 'cotton', 'grapes', 'jute', 'lentil', 'maize', 'mango', 'mothbeans', 'muskm  
Number of unique crops: 18

final\_df

	Element	Value	N	P	K	temperature	humidity	ph	rainfall	crop
200	Production	1050000.0	24	128	196	22.750888	90.694892	5.521467	110.431786	apple
201	Production	1050000.0	7	144	197	23.849401	94.348150	6.133221	114.051249	apple
202	Production	1050000.0	14	128	205	22.608010	94.589006	6.226290	116.039659	apple
203	Production	1050000.0	8	120	201	21.186674	91.134357	6.321152	122.233323	apple
204	Production	1050000.0	20	129	201	23.410447	91.699133	5.587906	116.077793	apple
...	...	...	...	...	...	...	...	...	...	...
129595	Production	3626000.0	97	12	47	25.287846	89.636679	6.765095	58.286977	watermelon
129596	Production	3626000.0	110	7	45	26.638386	84.695469	6.189214	48.324286	watermelon
129597	Production	3626000.0	96	18	50	25.331045	84.305338	6.904242	41.532187	watermelon
129598	Production	3626000.0	83	23	55	26.897502	83.892415	6.463271	43.971937	watermelon
129599	Production	3626000.0	120	24	47	26.986037	89.413849	6.260839	58.548767	watermelon

43200 rows × 10 columns

```
print("Null counts:")
print(final_df.isna().sum())
```

```
Null counts:
Element      0
Value        0
N            0
P            0
K            0
temperature  0
humidity     0
ph           0
rainfall     0
crop         0
dtype: int64
```

```
print("Duplicate rows:", final_df.duplicated().sum())
final_df = final_df.drop_duplicates().reset_index(drop=True)
print("Shape after removing duplicates:", final_df.shape)
```

```
Duplicate rows: 600
Shape after removing duplicates: (42600, 10)
```

```
numeric_cols = ["Value", "N", "P", "K", "temperature", "humidity", "ph", "rainfall"]
```

```
for col in numeric_cols:
    Q1 = final_df[col].quantile(0.25)
    Q3 = final_df[col].quantile(0.75)
    IQR = Q3 - Q1
    low = Q1 - 1.5 * IQR
    high = Q3 + 1.5 * IQR
    final_df = final_df[(final_df[col] >= low) & (final_df[col] <= high)]
```

```
print("Shape after outlier removal:", final_df.shape)
```

```
Shape after outlier removal: (31037, 10)
```

```
print("Unique crops:", sorted(final_df["crop"].unique()))
print("Number of unique crops:", final_df["crop"].nunique())
print(final_df["crop"].value_counts())
```

```
Unique crops: ['banana', 'chickpea', 'coconut', 'coffee', 'cotton', 'jute', 'lentil', 'maize', 'mango', 'mothbeans', 'muskmelon']
Number of unique crops: 15
crop
coconut      2400
jute          2400
lentil        2400
watermelon   2400
cotton        2400
coffee        2300
muskmelon    2300
```

```
mango      2232
chickpea   2088
pigeonpeas 2016
maize       2000
banana     1700
orange      1617
papaya      1488
mothbeans   1296
Name: count, dtype: int64
```

```
final_df = final_df.drop(columns=["Element"], errors="ignore")
```

```
numeric_features = ['N','P','K','temperature','humidity','ph','rainfall']
ranges = {feature: (final_df[feature].min(), final_df[feature].max()) for feature in numeric_features}
print(ranges)
```

```
{"N": (0, 140), 'P': (5, 95), 'K': (5, 85), 'temperature': (16.39624284, 36.32268069), 'humidity': (14.25803981, 99.98187601), 'ph': (5.4, 8.4), 'rainfall': (0.0, 490.0)}
```

```
final_df.to_csv("Clean_dataset_15.csv", index=False)
print("Dataset saved successfully")
```

```
Dataset saved successfully
```

```
print("Rows:", final_df.shape[0])
print("Columns:", final_df.shape[1])
```

```
Rows: 31037
Columns: 9
```

```
print("Null counts:")
print(final_df.isna().sum())
```

```
Null counts:
Value      0
N          0
P          0
K          0
temperature 0
humidity    0
ph          0
rainfall    0
crop        0
dtype: int64
```

```
# =====
# 1. IMPORT LIBRARIES
# =====
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from xgboost import XGBRegressor
```

```
# =====
# 2. BASIC CLEANING
# =====
final_df = final_df.drop_duplicates()
final_df = final_df.dropna()

num_cols = ["N", "P", "K", "temperature", "humidity", "ph", "Value"]
final_df[num_cols] = final_df[num_cols].apply(pd.to_numeric)
```

```
# =====
# 3. FEATURE ENGINEERING (IMPORTANT FOR 95%+)
# =====
final_df["NPK_sum"] = final_df["N"] + final_df["P"] + final_df["K"]
```

```
final_df["NP_ratio"] = final_df["N"] / (final_df["P"] + 1)
final_df["NK_ratio"] = final_df["N"] / (final_df["K"] + 1)
final_df["PK_ratio"] = final_df["P"] / (final_df["K"] + 1)

final_df["temp_humidity"] = final_df["temperature"] * final_df["humidity"]
final_df["temp_ph"] = final_df["temperature"] * final_df["ph"]
```

```
# =====
# 4. ENCODE CROP (INPUT FEATURE)
# =====
encoder = LabelEncoder()
final_df["Crop_enc"] = encoder.fit_transform(final_df["crop"])
```

```
# =====
# 5. DEFINE X & y
# Target = Production (Value)
# Element is DROPPED
# =====
X = final_df.drop(columns=["Value", "Element", "crop"], errors="ignore")
y = final_df["Value"]
```

```
# =====
# 6. SCALE FEATURES
# =====
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# =====
# 7. TRAIN-TEST SPLIT
# =====
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.20, random_state=42
)
```

```
# =====
# 8. HIGH-ACCURACY XGBOOST
# =====
xgb = XGBRegressor(
    n_estimators=700,
    learning_rate=0.04,
    max_depth=8,
    min_child_weight=1,
    subsample=0.9,
    colsample_bytree=0.9,
    gamma=0,
    reg_alpha=0.15,
    reg_lambda=1.1,
    objective="reg:squarederror",
    random_state=42,
    n_jobs=-1
)
```

```
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.85, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=0.1, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=7,
             max_leaves=None, min_child_weight=2, missing=np.nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=600,
             n_jobs=-1, num_parallel_tree=None, ...)
```

```
y_train_pred = xgb.predict(X_train)
y_test_pred = xgb.predict(X_test)

train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Train Accuracy (R²): {train_r2 * 100:.2f}%")
print(f"Test Accuracy (R²): {test_r2 * 100:.2f}%")
```

Train Accuracy (R<sup>2</sup>): 85.17%  
 Test Accuracy (R<sup>2</sup>): 83.80%

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=42)
r2_scores = []

for train_idx, val_idx in kf.split(X_scaled):
    X_tr, X_val = X_scaled[train_idx], X_scaled[val_idx]
    y_tr, y_val = y.iloc[train_idx], y.iloc[val_idx]

    model = XGBRegressor(
        n_estimators=600,
        learning_rate=0.05,
        max_depth=7,
        min_child_weight=2,
        subsample=0.85,
        colsample_bytree=0.85,
        gamma=0.1,
        reg_alpha=0.2,
        reg_lambda=1.2,
        objective="reg:squarederror",
        random_state=42,
        n_jobs=-1
    )

    model.fit(X_tr, y_tr)
    preds = model.predict(X_val)
    r2_scores.append(r2_score(y_val, preds))

print("Average CV R²:", np.mean(r2_scores) * 100)
```

Average CV R²: 83.4202081212002