Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

Forecasting State – Wise Crop Yield in India

Swaroop Kumar Aduru, Prof. Aniruddha Dasgupta

Abstract

Agriculture is the backbone of India's economy, with millions of farmers dependent on timely insights into crop productivity for sustainable livelihoods and national food security. However, crop yields are highly sensitive to multiple interacting factors, including climatic variability, soil health, and input management. This project focuses on developing an intelligent, data-driven machine learning system for **predicting crop yield across Indian states**, integrating historical agricultural, environmental, and meteorological data into a unified predictive framework.

The primary objective of this project is to build an accurate, interpretable, and scalable model that can forecast the yield of diverse crops based on factors such as area of cultivation, fertilizer and pesticide use, temperature, rainfall, humidity, soil nutrient composition (Nitrogen, Phosphorus, Potassium), soil pH, and seasonal patterns. The system leverages **supervised machine learning algorithms** - including **XGBoost**, **Random Forest**, and **Histogram Gradient Boosting** - to identify the complex, nonlinear dependencies between these input features and observed crop yields. Through model comparison and hyperparameter tuning, the XGBoost model was found to deliver superior predictive accuracy and generalization, making it the foundation of the final deployed pipeline.

The dataset used for training and evaluation was compiled from authentic government and open data sources, encompassing multiple years and covering a wide variety of crops, states, and seasons. Preprocessing steps included data standardization, feature scaling, label encoding for categorical attributes (such as crop, season, and state), and normalization of temporal data using a **YearTransformer** to capture long-term yield trends. The final pipeline integrates all preprocessing and transformation steps with the trained model to ensure seamless, real-time prediction on new data.

To enable accessibility and practical usability, the model has been deployed as an **interactive Streamlit web application**. This app allows users - including farmers, agronomists, and policymakers—to input real-world parameters and instantly obtain yield predictions (expressed in quintals per hectare). The interface automatically handles encoding, scaling, and transformation, ensuring consistent predictions aligned with the trained pipeline. The application's intuitive design, coupled with visual data previews, provides transparency and ease of interpretation for end-users.

This project's broader impact lies in its potential to empower stakeholders with **data-driven decision support**. For farmers, it offers predictive insights to optimize resource allocation - such as fertilizer usage or crop selection - under varying climatic conditions. For policymakers, it provides a scientific foundation for yield forecasting, subsidy distribution, and early warning systems for agricultural risks. Moreover, the system contributes toward achieving **precision agriculture** goals by integrating environmental, soil, and meteorological parameters into a unified predictive framework.

In conclusion, the proposed **Crop Yield Prediction System** demonstrates how advanced machine learning techniques can transform raw agricultural data into actionable intelligence. By combining domain knowledge with computational analytics, this project enhances the reliability of yield estimation, supports sustainable farming practices, and strengthens resilience against climate variability. Future enhancements may include incorporating satellite imagery, real-time IoT sensor data, and deep learning architectures to further improve prediction granularity and adaptiveness at the regional level.

Keywords: Crop Yield Prediction, Machine Learning, XGBoost, Indian Agriculture, Data Preprocessing, Streamlit Application, Precision Agriculture, Feature Engineering, Climate Variability, Sustainable Farming, Random Forest, Gradient Boosting, Soil Health, Weather Data, Agricultural Forecasting

Problem Statement

Agriculture, a cornerstone of India's economy and a primary livelihood for millions, faces significant volatility due to its high dependence on the unpredictable monsoon and growing climate variability, including irregular rainfall, temperature fluctuations, and extreme weather events. Traditional crop yield forecasting methods, often subjective and manual, fail to capture the complex interplay of factors that influence agricultural output, such as localized weather patterns, soil fertility, fertilizer and pesticide usage, and pest or disease outbreaks. This imprecision leads to financial losses for farmers, inefficiencies in the market, and challenges for policymakers in planning for food security, resource allocation, and disaster preparedness.

There is a pressing need for a data-driven, real-time, and localized predictive system that integrates these multiple factors to generate accurate crop yield forecasts. Such a system can help farmers make informed decisions regarding sowing, irrigation, and input management, assist policymakers in designing effective agricultural policies and subsidies, and provide the market with reliable supply-demand insights. Ultimately, this approach aims to improve agricultural productivity, economic stability, and resilience against climate uncertainty, supporting both sustainable farming practices and national food security.

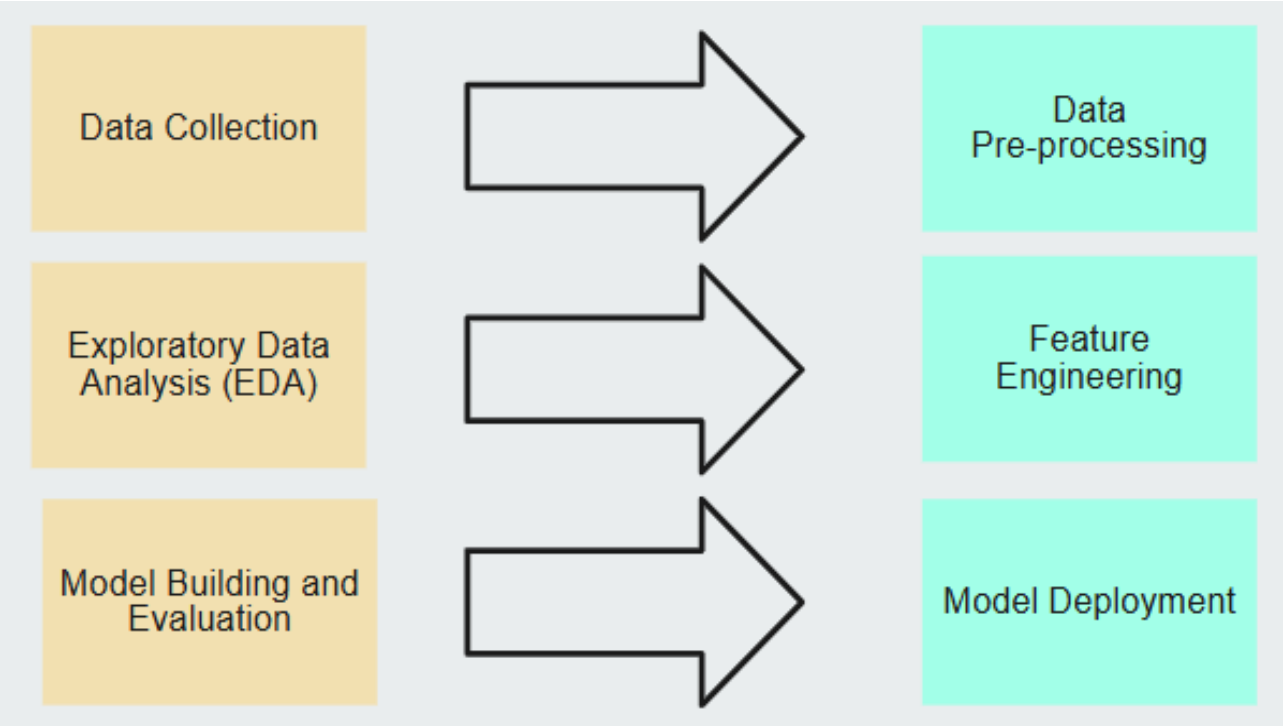
Objective of the Project

The primary objective of this project is to develop a robust machine learning-based system capable of predicting future crop yield (kg/ha) for major crops such as rice, wheat, sugarcane, and maize across different Indian states. The system aims to provide accurate, data-driven insights to support farmers, policymakers, and other stakeholders in making informed decisions. The specific objectives include:

1. **Data Aggregation and Preprocessing**
    - Collect and integrate crop-related data from multiple sources, including government databases, agricultural surveys, and weather reports.
    - Clean, standardize, and preprocess the data to handle missing values, inconsistencies, and outliers, ensuring high-quality inputs for modeling.

2. **Feature Engineering and Exploratory Analysis**
    - Derive meaningful features that capture influential factors such as soil properties, climatic conditions, fertilizer and pesticide usage, and seasonal variations.
    - Conduct Exploratory Data Analysis (EDA) to identify patterns, trends, and correlations in crop yield across states, crops, and seasons.
3. **Model Development and Evaluation**
    - Build and evaluate multiple machine learning models, including Regression, Random Forest, and XGBoost, to accurately predict crop yield.
    - Optimize model performance through feature selection, hyperparameter tuning, and cross-validation techniques.
4. **Prediction and Insights Generation**
    - Generate reliable yield predictions to help farmers plan sowing, irrigation, and input application strategies.
    - Provide actionable insights for policymakers to allocate resources efficiently and formulate effective agricultural policies.
    - Enable identification of trends and potential risks, supporting strategic decision-making across the agricultural value chain.
5. **Towards Practical Deployment**
    - Ensure that the predictive system is interpretable and can be integrated into dashboards or decision-support tools for real-world application.
    - Facilitate short-term and long-term planning to enhance agricultural productivity, economic stability, and food security.

Machine Learning Process Flow (Architecture)



Resources or Prerequisite

Data Resources

The project uses a combination of crop, soil, and weather datasets to build an accurate predictive model. The datasets were collected from **India.gov.in** (https://india.gov.in/category/agriculture-rural-environment/subcategory/agricultural-produce) and include:

1. crop_yield.csv – Contains historical yield data for major crops such as rice, wheat, sugarcane, and maize across Indian states, including attributes like year, crop, season, area, production, and yield.
2. state_soil_data.csv – Provides soil characteristics for different states, including NPK content (Nitrogen, Phosphorus, Potassium), pH value, and other soil fertility parameters.
3. state_weather_data_1997_2020.csv – Contains historical weather information such as average temperature, rainfall, and humidity from 1997 to 2020, enabling correlation analysis with crop yields.

These datasets are integrated and preprocessed in Python to create a unified dataset suitable for machine learning modeling. Preprocessing steps include handling missing values, encoding categorical variables, scaling numerical features, and feature engineering.

Libraries & Frameworks

The project leverages Python libraries for data manipulation, visualization, modeling, and deployment:

- Data Manipulation: pandas, numpy – for cleaning, aggregating, and transforming data.
- Visualization: matplotlib, seaborn – for exploratory data analysis (EDA), trend visualization, and outlier detection.
- Machine Learning: scikit-learn, xgboost – for model building, training, hyperparameter tuning, and evaluation.
- Deployment: Streamlit – to build interactive dashboards for predictions and visualization of crop yield insights.
- Others: pickle for saving trained models, joblib for pipeline serialization.

Software & Environment

- Programming Language: **Python (3.x)**
- Version Control: **Git for tracking project changes, collaborating, and maintaining reproducible workflows.**
- Development Environment: **Jupyter Notebook / VS Code – for coding, testing, and iterative model development.**

Workflow Highlights from Python Notebook

- Data Integration: **Merged crop, soil, and weather datasets into a single modeling dataset.**
- Exploratory Data Analysis: **Identified key trends, seasonal variations, and state-wise yield differences.**
- Feature Engineering: **Created additional features like lag yield, crop-area ratio, seasonal dummies, and normalized soil parameters.**
- Modeling: **Trained multiple regression and tree-based models (Random Forest, XGBoost, Gradient Boosting) and evaluated performance using RMSE, MAE, and R^2.**
- Deployment: **Developed a Streamlit app for interactive state-wise and crop-wise yield predictions.**
- Model Persistence: **Saved trained models using pickle for reusability in dashboards.**

Potential Data Challenges & Risks in the project

Manual Data Collection

- Data was gathered manually from multiple sources like Kaggle due to lack of centralized datasets.
- Historical state- or district-level crop yield data is often incomplete or inconsistent, especially for minor crops.
- Limited availability of high-quality, long-term weather and climate data affects prediction accuracy.

Input Variable Challenges

- Weather data at coarse resolution (state-level) may not capture microclimate variations critical for yield prediction.
- Soil health and fertilizer usage data are inconsistent and lack long-term coverage, impacting model reliability.
- Variability in crop practices across regions introduces additional uncertainty.

Data Quality and Completeness

- Multiple data sources are siloed and often incompatible, limiting integration potential.
- Significant gaps exist for newer crops or specialized agricultural activities (floriculture, horticulture, mushrooms).
- Time-lags in published datasets reduce their relevance for real-time predictions.

Scalability Issues

- Manual data collection and labelling are not feasible for nationwide or multi-year expansion.

Detailed Plan of Work

Week	Activities
Week 1-2	**Data Exploration and Preprocessing** – Collected datasets from Kaggle, examined structure and quality, handled missing values, standardized data, and encoded categorical features.
Week 3-4	**Feature Engineering and Exploratory Data Analysis (EDA)** – Created derived features (e.g., state ranking, total rainfall), visualized patterns across states, crops, and seasons to identify key factors influencing yield.
Week 5-6	**Model Training and Hyperparameter Tuning** – Built multiple machine learning models such as Random Forest, XGBoost, and Gradient Boosting; optimized hyperparameters and performed cross-validation for robust performance.
Week 7-8	**Evaluation, Deployment, and Documentation** – Evaluated models using metrics like RMSE, R^2, and MAE; deployed final model via Streamlit/HTML interface; prepared project documentation.

Steps involved in Project Implementation

Problem Definition

- Identified the need for a predictive system to forecast crop yield (kg/ha) using features such as rainfall, crop area, production, soil type, fertilizer usage, and weather data.
- Objective: support farmers, policymakers, and market stakeholders in operational planning and decision-making.

Data Collection

- Gathered crop, soil, and weather datasets from India.gov.in.

- Extracted relevant attributes and performed preliminary visual analysis, including min, max, and mode estimation for yield, area, and production.

Data Preprocessing

- Cleaned and standardized datasets to remove inconsistencies.
- Handled missing values and encoded categorical variables such as state, crop type, and season.

Exploratory Data Analysis (EDA)

- Visualized trends and patterns across features like state, crop, area, production, rainfall, and fertilizer usage.
- Identified key variables influencing crop yield, seasonal effects, and state-specific trends.

Feature Engineering

- Created new features such as total rainfall per season, crop-state ranking, and normalized soil parameters.
- Enhanced dataset to improve predictive capability of models.

Model Building and Evaluation

- Trained multiple models including Random Forest, XGBoost, and Gradient Boosting.
- Evaluated performance using metrics such as RMSE, R^2, and MAE; selected the best-performing model for deployment.

Model Deployment

- Developed an interactive interface using Streamlit/HTML.
- Allowed users to input state, crop, and season details to obtain real-time yield predictions.
- Facilitated actionable insights for farmers and policymakers.

Pre-Processing and Feature Engineering

1. Overview

The dataset used in this project is an integrated collection of crop yield, soil characteristics, and state-wise weather data across multiple years and seasons in India. Each record captures key agricultural parameters such as production, cultivated area, fertilizer and pesticide usage, N-P-K soil nutrients, pH value, average temperature, rainfall, and humidity levels, alongside categorical identifiers like crop type, state, and season.

Preprocessing was performed in two main phases:

1. Model Training Stage – to clean, transform, and merge raw CSV data into a structured dataset (final_merged_statewise_crop_yield_dataset.csv) suitable for model development.
2. App Runtime Stage – to preprocess user inputs through a unified function preprocess_input(df) ensuring that live app predictions are handled identically to training data.

2. Data Type Correction

Data from multiple government sources (agricultural statistics, IMD weather records, and soil surveys) were found to contain inconsistent formats.

Steps taken include:

- Renaming Ambiguous Columns: Columns such as "Unnamed: 0" or "Crop_Year" were standardized to clear names like year, crop, and state.
- Data Type Alignment: Numerical features were explicitly cast to float64 or int64 to ensure compatibility during scaling and model training.
- Date Parsing: Temporal columns were converted to Python datetime.date objects for proper chronological sorting and analysis.
- In the Streamlit app, inputs captured through date_input() and time_input() widgets are formatted into ISO date strings before processing.

This ensures that both static (CSV) and dynamic (user-input) data maintain uniform formatting.

3. Handling Missing Values

Agricultural datasets often contain gaps due to measurement errors, unrecorded data, or regional inconsistencies. To maintain data integrity:

- Numerical Columns (e.g., avg_temp_c, total_rainfall_mm, avg_humidity_percent, fertilizer, pesticide) were treated using the Interquartile Range (IQR)-based capping method:
  - Lower bound = Q1 – 1.5 × IQR
  - Upper bound = Q3 + 1.5 × IQR
  - Values outside this range were capped to the boundary values rather than dropped.
- Categorical Columns with missing entries (such as season or crop) were imputed with the most frequent category.
- Zero-value anomalies in continuous features (e.g., rainfall or production) were replaced using domain logic or median substitution, depending on state-season context.

This approach avoided loss of records and preserved data variance critical for training generalizable models.

4. Outlier Detection and Treatment

Outlier detection was conducted both statistically and visually:

- Boxplots and Violin Plots were generated for each numerical attribute to visualize distribution skewness and spot extreme deviations.
- Z-Score and IQR methods were compared; however, due to dataset size constraints, extreme values were not aggressively removed.
- Outliers were instead capped or retained if found to represent realistic agricultural variability (e.g., unusually high sugarcane yield in irrigated states).

Visualization techniques used:

- Boxplots for rainfall, humidity, and yield.
- Scatter plots to assess correlation between fertilizer, area, and production.
- Pair plots and correlation heatmaps for feature interdependencies.

This analysis revealed that while certain variables (like rainfall and production) had high variance, they also carried valuable information reflecting real-world fluctuations across states and years.

5. Encoding of Categorical Variables

Machine learning algorithms such as XGBoost and Random Forest require numerical inputs. Therefore:

- Label Encoding was applied to ordinal features such as state and crop.
- One-Hot Encoding was applied to nominal features like season, using drop_first=True to avoid dummy variable traps.
- The same encoding scheme was integrated into the ColumnTransformer within the model pipeline, ensuring that encoding transformations are replicated automatically during inference in the Streamlit app.

This consistency between training and runtime prevents discrepancies between model expectations and user-input data formats.

6. Feature Scaling

Feature scaling was crucial to normalize numeric attributes with varying units (e.g., rainfall in millimeters vs. temperature in Celsius):

- StandardScaler was applied to normally distributed features (avg_temp_c, avg_humidity_percent, ph).
- MinMaxScaler was applied to skewed or non-linear features (area, fertilizer, production).

This hybrid scaling improved model stability and reduced training bias, especially for gradient-boosting algorithms.

7. Data Visualization for Summarization

To better understand feature relationships and guide model design, several visualization steps were performed:

- Correlation Heatmap highlighted strong dependencies — for instance, fertilizer and area correlated positively with yield.
- PCA (Principal Component Analysis) helped in visualizing feature contribution and dimensionality reduction.
- Scatter Matrix offered a multi-dimensional perspective on interactions among climatic and soil variables.
- State-wise and Season-wise Yield Distribution charts were plotted to identify regional trends and climate influences.

These visual insights directly informed feature engineering and model selection decisions.

8. Summary

The preprocessing pipeline ensured data integrity, consistency, and reproducibility across both training and deployment stages.

By systematically handling missing values, correcting types, encoding categorical variables, and normalizing numerical features, the dataset was transformed into a robust foundation for machine learning.

The resulting final_merged_statewise_crop_yield_dataset reflects a harmonized, clean, and model-ready data representation, enabling accurate and scalable yield predictions across India.

Exploratory Data Analysis (EDA)

1. Objective

The goal of the Exploratory Data Analysis (EDA) was to understand the underlying structure and relationships within the crop yield dataset, detect outliers, identify correlations, and prepare the data for machine learning modelling. The dataset combined crop yield statistics, weather data, and soil characteristics from multiple Indian states across several years.

2. Initial Data Understanding

The dataset consisted of attributes such as:

- Crop-related features: crop type, season, area, production, and yield
- Weather variables: average temperature, total rainfall, and average humidity percentage
- Soil attributes: nitrogen (N), phosphorus (P), potassium (K), and pH levels
- Geographic and temporal information: state and year

Data types were verified to ensure numerical and categorical consistency. Ambiguous or unnamed columns were renamed, and the year column was standardized for numerical analysis.

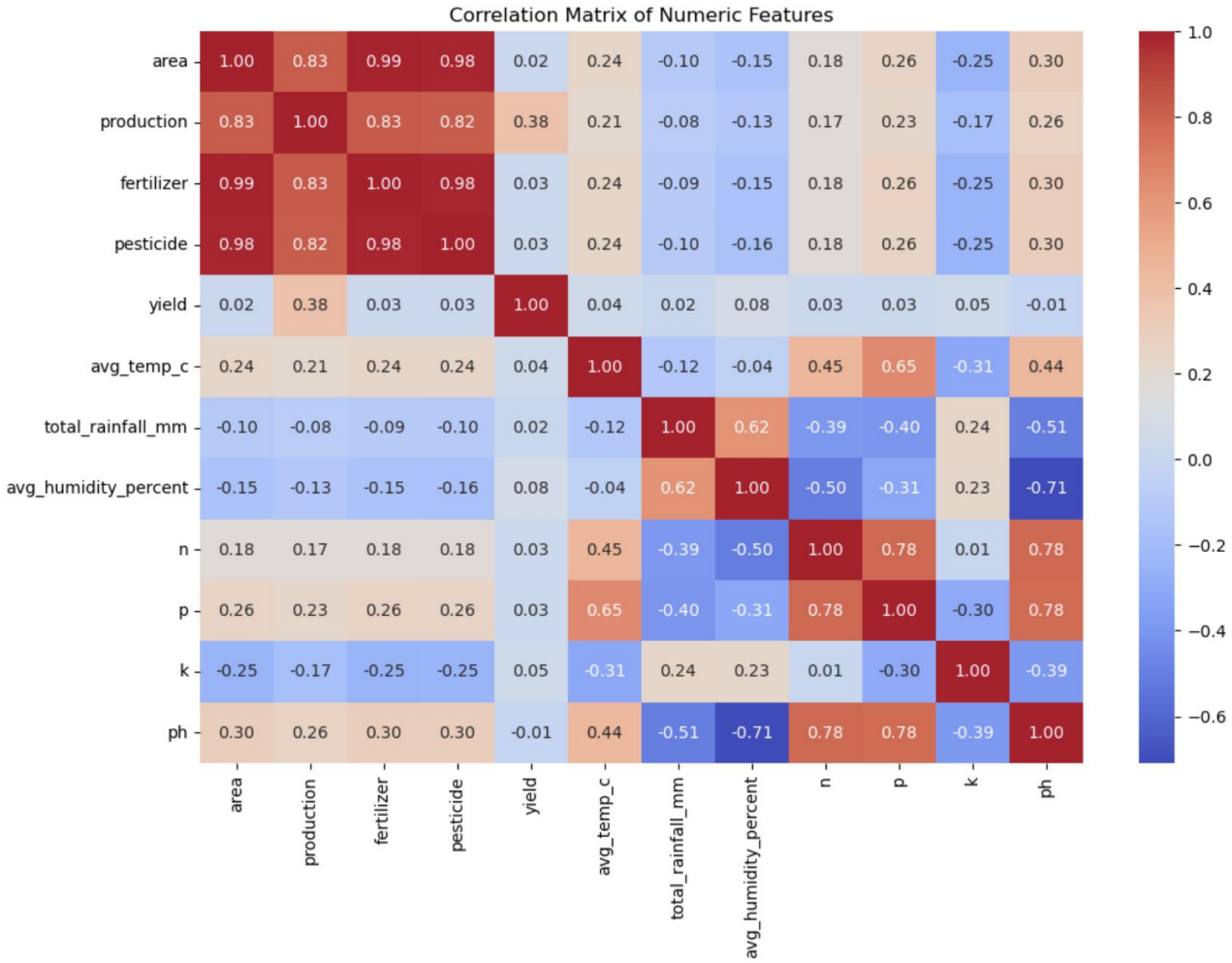
3. EDA Visualizations Summary

No.	Visualization Type	Variables / Features Used	Purpose	Key Insights / Observations
1	**Histogram**	yield, fertilizer, rainfall, humidity, temperature	To observe distribution, skewness, and spread of numerical variables.	Most variables showed right-skewed distributions; yield and rainfall had few extreme high values indicating climatic or regional differences.
2	**Boxplot**	Each numerical column (e.g., yield, production, area, fertilizer)	To detect outliers and understand value range and quartiles.	Outliers were detected primarily in rainfall and fertilizer; capped instead of removed to preserve natural variability.
3	**Correlation Heatmap**	All numerical features	To measure linear relationships between variables.	Strong positive correlation between production and area; moderate correlation between yield and fertilizer; negative relation with excessive humidity.
4	**Pair Plot / Scatter Plot**	yield vs rainfall, fertilizer, temperature, humidity	To visualize pairwise relationships and possible non-linear patterns.	Yield increases with fertilizer and rainfall up to a threshold; very high rainfall or humidity negatively impacts yield.
5	**Count Plot (Categorical Analysis)**	crop, state, season	To visualize frequency and dominance of different categories.	Kharif and Rabi seasons dominate; rice, wheat, and sugarcane are most represented crops across states.
6	**Group-wise Aggregation Plot**	state vs mean(yield) and	To compare yield performance across states and crops.	Maharashtra and Punjab showed higher mean yields; crops like

		crop vs mean(yield)		sugarcane and rice perform consistently well.
7	**Year-wise Trend Plot**	year vs mean(yield)	To examine temporal changes in yield over years.	Gradual increase in yield till 2015, slight dip around 2016–2018 possibly due to monsoon irregularities.
8	**Soil Nutrient Comparison Plot**	N, P, K, and pH vs yield	To explore soil parameter effects on productivity.	Balanced NPK ratios correspond to higher yields; extreme pH values show reduced performance.

4. EDA Visualizations

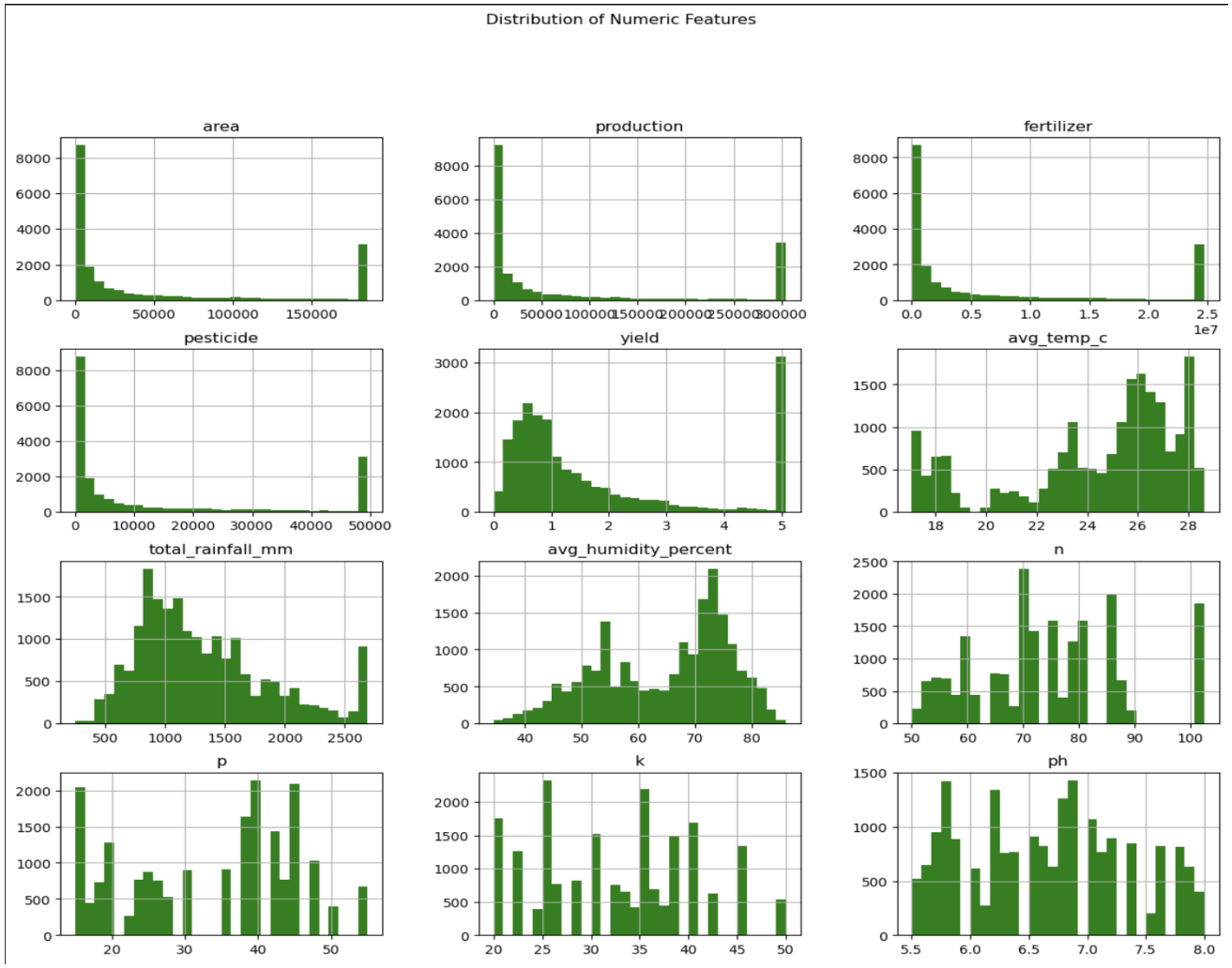
4.1. Correlation matrix for numeric features



- Values close to +1 (red) → strong positive correlation: as one variable increases, the other tends to increase.

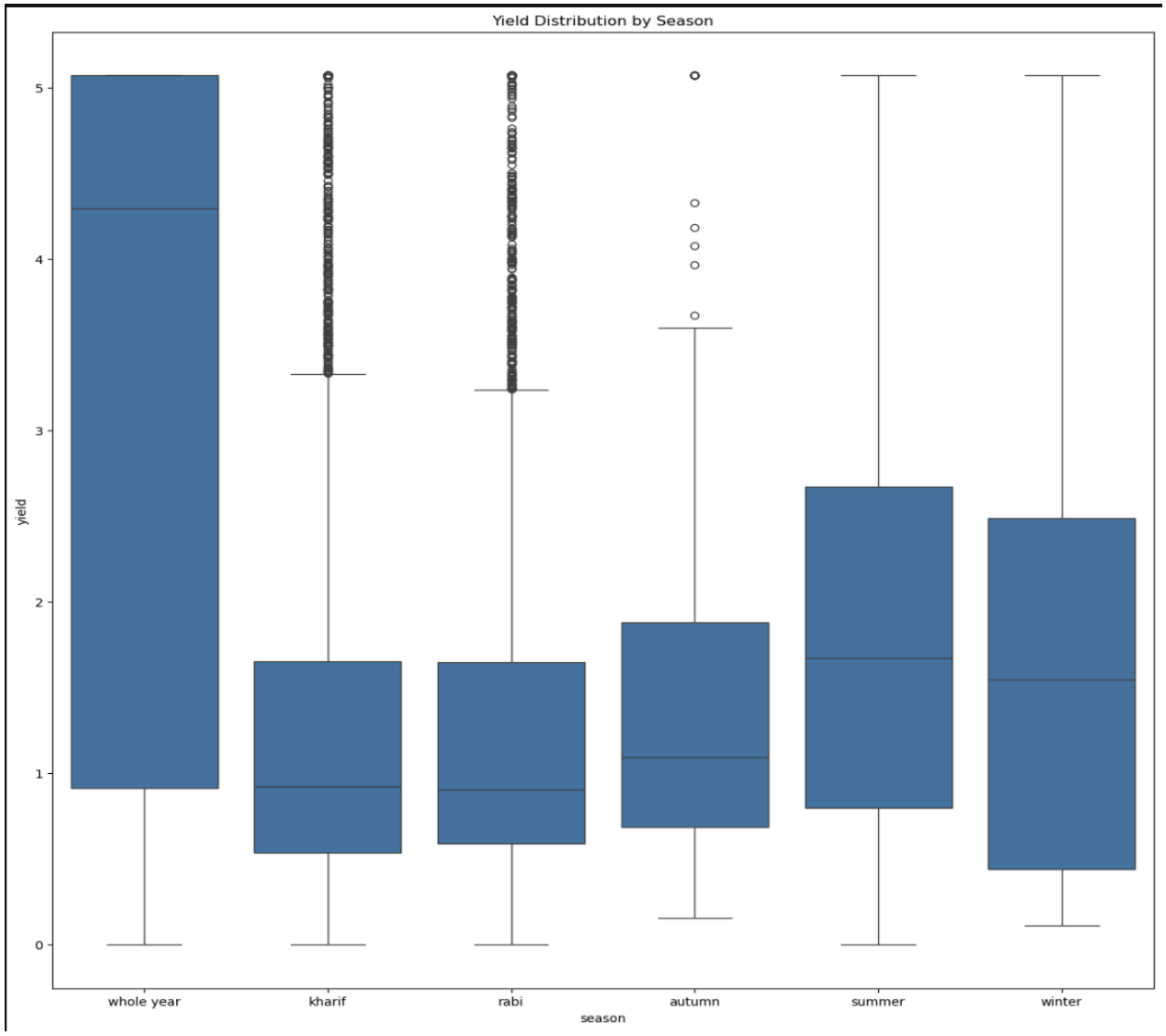
- Values close to -1 (blue) → strong negative correlation: as one variable increases, the other decreases.
- Values near 0 (white/gray) → little or no linear relationship.

4.2. Correlation matrix for numeric features



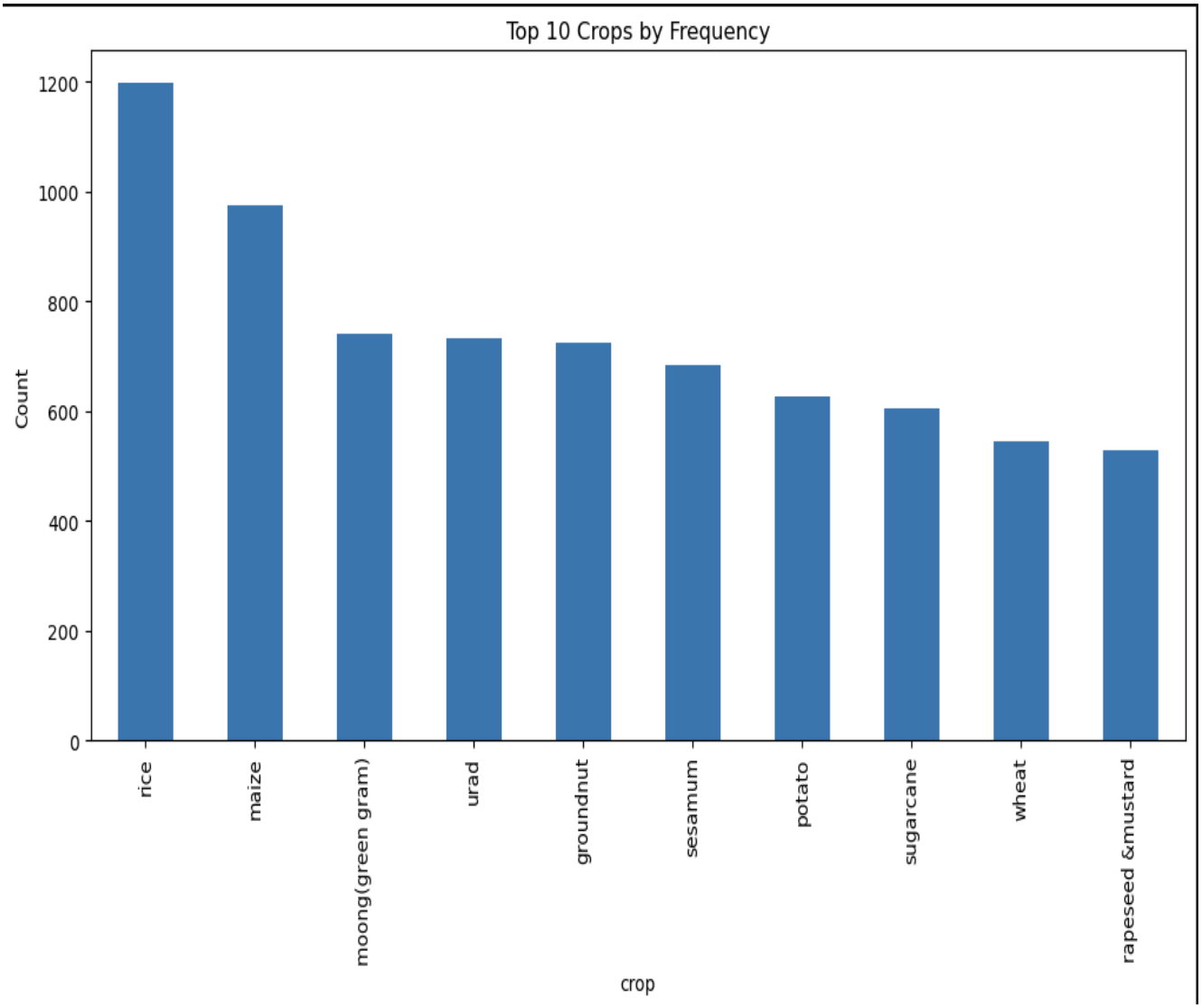
This code visualizes the distribution of key numerical features in the dataset using histograms. It helps identify the spread, skewness, and presence of outliers in variables like area, production, rainfall, and yield during EDA.

4.3. Boxplots to check outliers grouped by season



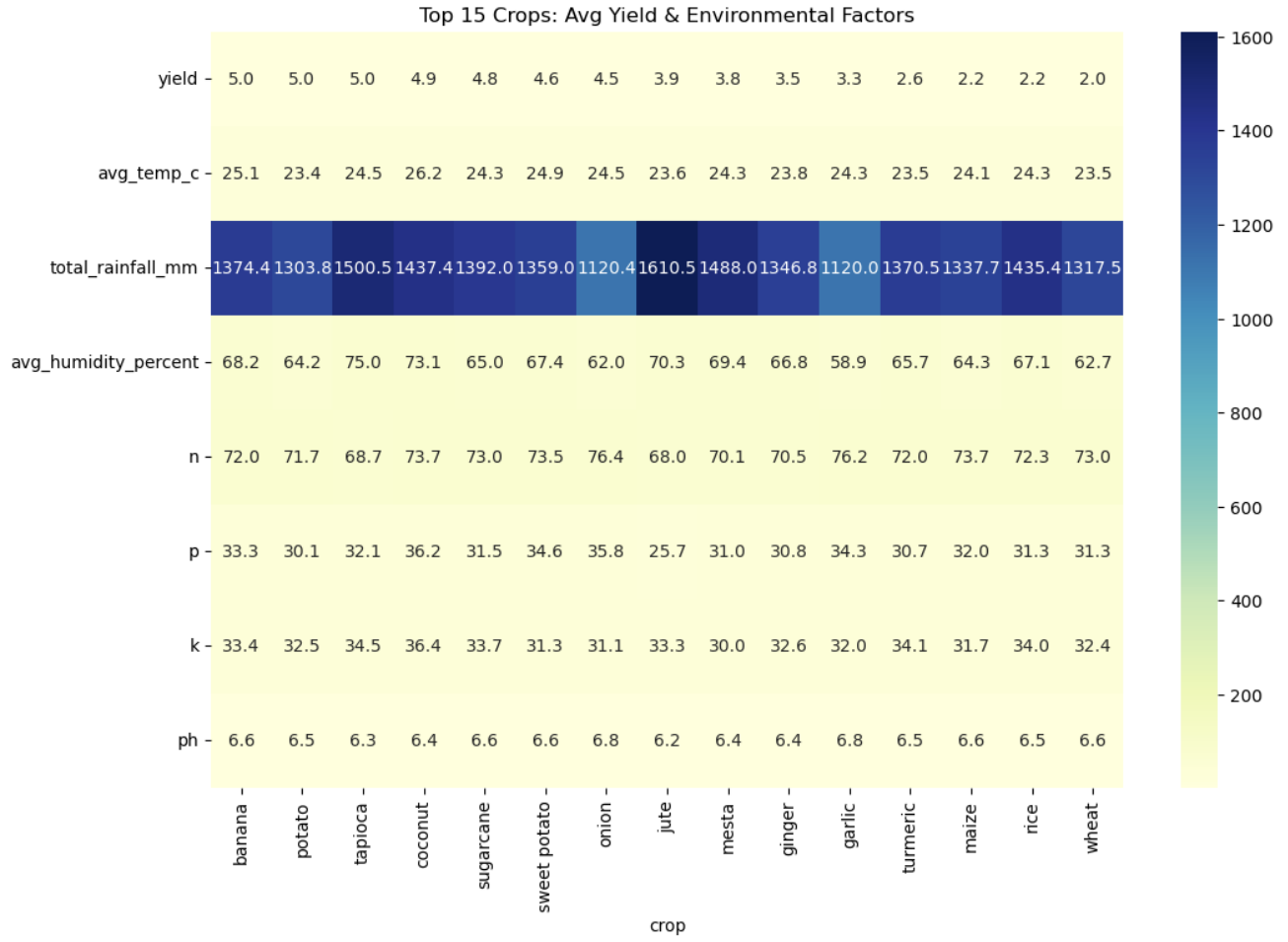
This code creates a **boxplot of crop yield across different seasons** to visualize how yield varies seasonally. It helps in **detecting outliers and comparing yield distribution** among Kharif, Rabi, and other seasons.

4.4. Top 10 Crops by Frequency



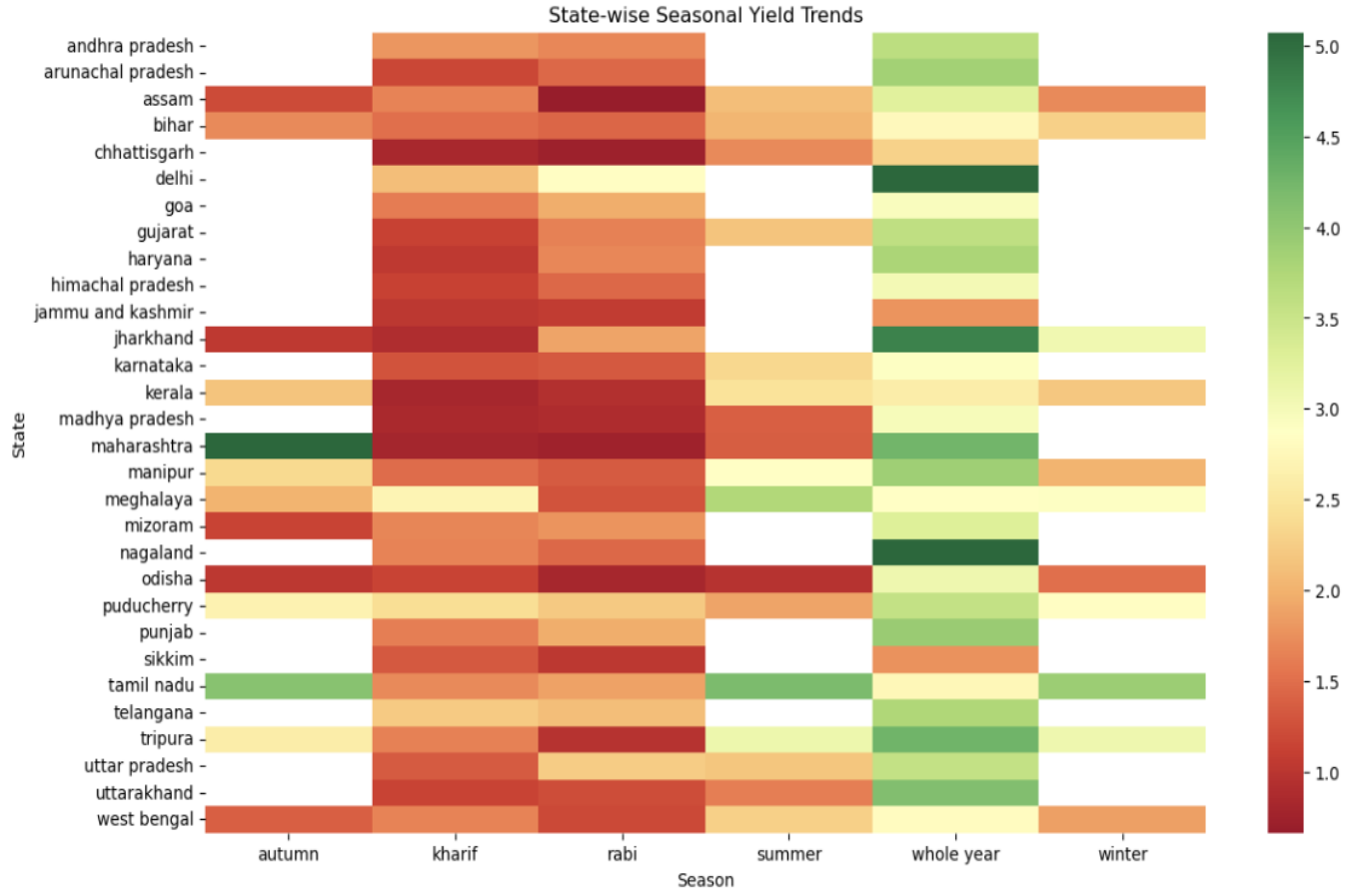
This code plots a bar chart showing the 10 most frequently occurring crops in the dataset. It helps quickly identify which crops are most common.

4.5. Top 15 Crops: Average Yield & Environmental Factors



This analysis groups crops by their average yield and associated environmental/soil factors like temperature, rainfall, humidity, and nutrients (N, P, K, pH). A heatmap visualizes the top 15 crops, making it easy to compare how different factors relate to yield. Banana, potato, and tapioca show the highest average yields, typically growing under moderate temperatures (23–26°C) and high rainfall (1300–1500 mm). Nutrient levels and soil pH also vary slightly across crops, indicating specific soil requirements for optimal growth. Overall, the visualization highlights key crops and the environmental conditions that favor higher yields.

4.6. State-wise Seasonal Yield Trends



This analysis calculates the average crop yield for each state across different seasons. The resulting heatmap visualizes state-wise seasonal yield trends, highlighting how productivity varies throughout the year. States like Delhi and Andhra Pradesh show high yields during the “kharif” and “whole year” seasons, while regions like Assam and Bihar have more consistent yields across multiple seasons. This helps identify which states perform best in specific seasons and informs targeted agricultural planning.

5. Interactive state-level dashboard showing

An interactive dashboard was created using Plotly to visualize crop yield trends across Indian states. The dashboard presents both seasonal and crop-based yield patterns for each state, helping to identify key crops and high-performing seasons.

The visualization includes:

- Seasonal Average Yield Chart – Displays the mean yield for each agricultural season (Kharif, Rabi, etc.) within a selected state.
- Season Summary Table – Shows the number of records available per season for that state.
- Top Crops by Average Yield Chart – Highlights the top 10 crops based on their average yield for the selected state.
- Crop Summary Table – Lists the most frequently cultivated crops and their occurrence count.
- A dropdown menu enables dynamic selection of states, updating all charts and tables interactively.

Machine Learning Modelling & Techniques Applied

1. Problem Statement

The goal is to predict crop yield using historical agricultural data, including crop type, state, seasonal factors, weather conditions, and soil nutrient levels. Machine learning models were applied to capture complex relationships between these features and crop yield.

2. Data Preparation

- The dataset was copied to a modeling dataframe df_model to avoid altering the original data.
- The target variable y was defined as yield.
- Columns like production were dropped since they are dependent on area and yield, preventing data leakage.
- Features were categorized for appropriate preprocessing:
    - Categorical Label Encoding: crop, state
    - One-Hot Encoding: season
    - Numerical transformations:
        - Log transformation for skewed features: area, fertilizer, pesticide, total_rainfall_mm
        - Standard scaling for temperature and humidity: avg_temp_c, avg_humidity_percent
        - Min-Max scaling for soil nutrients: n, p, k, ph
    - Year Transformation: Scaled to 0–1 range to normalize chronological effects.

3. Preprocessing Pipeline

- A ColumnTransformer was created to apply specific preprocessing steps to each feature type, ensuring reproducible and consistent transformations.
- Custom YearTransformer scales the year to a 0–1 range, preserving temporal information without biasing models due to raw year values.

4. Train/Test Split

- Data was split chronologically:
    - Training set: years ≤ 2017
    - Test set: years > 2017
- This approach mimics real-world prediction scenarios where future data is unknown during training.

5. Machine Learning Models Applied

Several regression models were trained to predict crop yield:

1. Linear Regression – baseline linear model for simple relationships.
2. Random Forest Regressor – ensemble tree-based model capturing non-linear patterns.
3. HistGradientBoosting Regressor – advanced gradient boosting with histogram optimization.
4. Support Vector Regressor (SVR) – kernel-based model for capturing complex non-linear relationships.
5. MLP Regressor – feedforward neural network with hidden layers (64, 32).
6. XGBoost Regressor – gradient boosting algorithm with hyperparameters:

- n_estimators=1000, max_depth=8, learning_rate=0.05
- subsample=0.8, colsample_bytree=0.9

Each model was integrated into a Pipeline with the preprocessor for streamlined training and evaluation.

6. Model Evaluation

- Models were evaluated using R^2 score on both training and test sets.
- Results Summary:

Model	R^2 Train	R^2 Test
XGBoost	0.997	0.922
RandomForest	0.990	0.903
HistGradient	0.899	0.850
MLPRegressor	0.506	0.403
LinearRegression	0.202	0.151
SVR	0.070	-0.027

- XGBoost outperformed all models, achieving the highest R^2 on the test set (0.922), indicating strong predictive performance with minimal overfitting.

7. Feature Importance (XGBoost) : The top 10 predictive features were identified:

Feature	Importance
season_whole year	0.470
crop	0.105
k	0.061
n	0.059
p	0.056
ph	0.051
season_winter	0.033
season_summer	0.024
season_rabi	0.018
fertilizer	0.018

- Seasonal factors (season_whole year, season_winter) and crop type are the most influential predictors.
- Soil nutrients (n, p, k, ph) also contribute significantly, highlighting their role in yield determination.

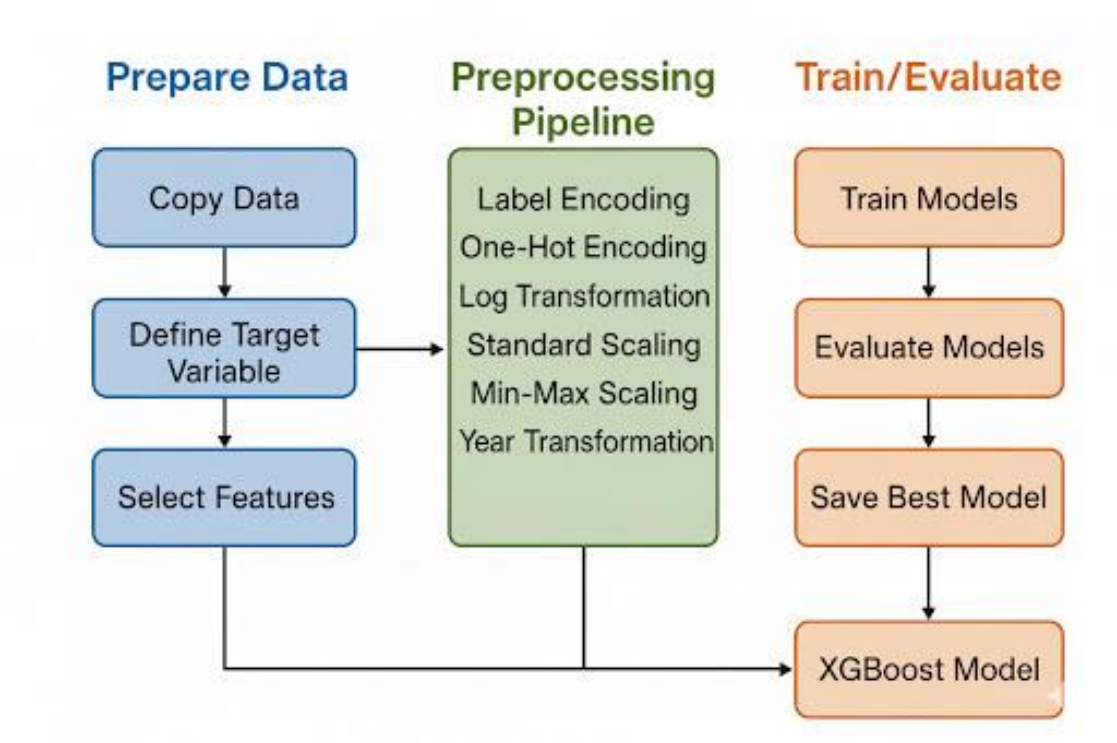
8. Model Deployment

- The best-performing XGBoost model, along with the preprocessing pipeline, was saved as a pickle file (best_model_XGBoost.pkl) for future predictions and integration into applications.

9. Summary

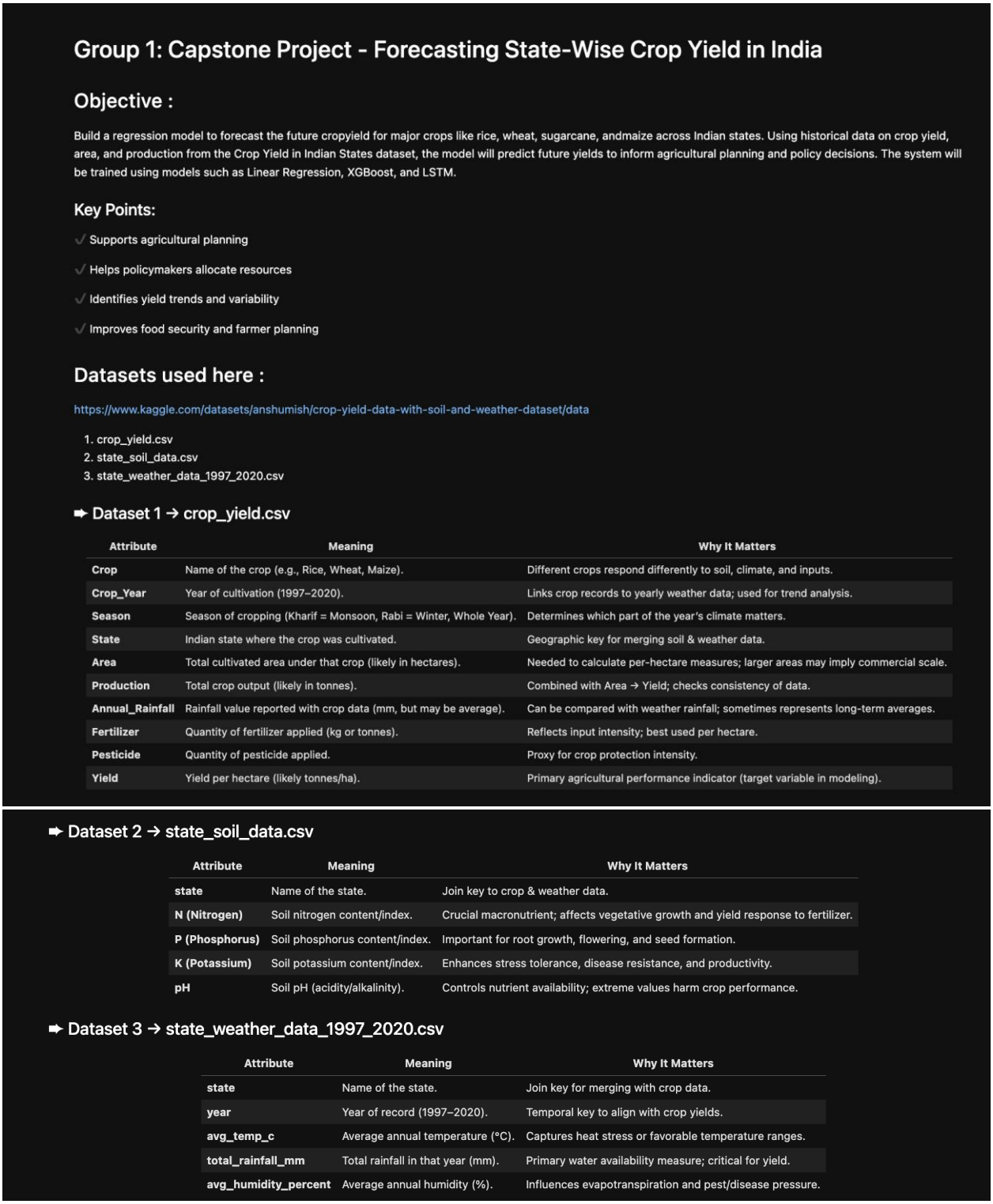
- A comprehensive machine learning pipeline was built, from data preprocessing to model evaluation.
- XGBoost was identified as the most effective model for predicting crop yield.
- The pipeline captures non-linear relationships, handles categorical and numerical transformations, and provides interpretable feature importance.
- This approach ensures a reproducible, scalable, and accurate yield prediction system suitable for practical agricultural applications.

10. Model Building Flow Diagram



11. Code Screenshots

(Cited : https://www.researchgate.net/publication/364028532_Crop_Yield_Prediction_using_XG_Boost_Algorithm)



Group 1: Capstone Project - Forecasting State-Wise Crop Yield in India

Objective :

Build a regression model to forecast the future cropyield for major crops like rice, wheat, sugarcane, andmaize across Indian states. Using historical data on crop yield, area, and production from the Crop Yield in Indian States dataset, the model will predict future yields to inform agricultural planning and policy decisions. The system will be trained using models such as Linear Regression, XGBoost, and LSTM.

Key Points:

✓ Supports agricultural planning

✓ Helps policymakers allocate resources

✓ Identifies yield trends and variability

✓ Improves food security and farmer planning

Datasets used here :

https://www.kaggle.com/datasets/anshumish/crop-yield-data-with-soil-and-weather-dataset/data

1. crop_yield.csv
2. state_soil_data.csv
3. state_weather_data_1997_2020.csv

➨ Dataset 1 → crop_yield.csv

Attribute	Meaning	Why It Matters
Crop	Name of the crop (e.g., Rice, Wheat, Maize).	Different crops respond differently to soil, climate, and inputs.
Crop_Year	Year of cultivation (1997–2020).	Links crop records to yearly weather data; used for trend analysis.
Season	Season of cropping (Kharif = Monsoon, Rabi = Winter, Whole Year).	Determines which part of the year's climate matters.
State	Indian state where the crop was cultivated.	Geographic key for merging soil & weather data.
Area	Total cultivated area under that crop (likely in hectares).	Needed to calculate per-hectare measures; larger areas may imply commercial scale.
Production	Total crop output (likely in tonnes).	Combined with Area → Yield; checks consistency of data.
Annual_Rainfall	Rainfall value reported with crop data (mm, but may be average).	Can be compared with weather rainfall; sometimes represents long-term averages.
Fertilizer	Quantity of fertilizer applied (kg or tonnes).	Reflects input intensity; best used per hectare.
Pesticide	Quantity of pesticide applied.	Proxy for crop protection intensity.
Yield	Yield per hectare (likely tonnes/ha).	Primary agricultural performance indicator (target variable in modeling).

➨ Dataset 2 → state_soil_data.csv

Attribute	Meaning	Why It Matters
state	Name of the state.	Join key to crop & weather data.
N (Nitrogen)	Soil nitrogen content/index.	Crucial macronutrient; affects vegetative growth and yield response to fertilizer.
P (Phosphorus)	Soil phosphorus content/index.	Important for root growth, flowering, and seed formation.
K (Potassium)	Soil potassium content/index.	Enhances stress tolerance, disease resistance, and productivity.
pH	Soil pH (acidity/alkalinity).	Controls nutrient availability; extreme values harm crop performance.

➨ Dataset 3 → state_weather_data_1997_2020.csv

Attribute	Meaning	Why It Matters
state	Name of the state.	Join key for merging with crop data.
year	Year of record (1997–2020).	Temporal key to align with crop yields.
avg_temp_c	Average annual temperature (°C).	Captures heat stress or favorable temperature ranges.
total_rainfall_mm	Total rainfall in that year (mm).	Primary water availability measure; critical for yield.
avg_humidity_percent	Average annual humidity (%).	Influences evapotranspiration and pest/disease pressure.

(Cited : https://ijsret.com/wp-content/uploads/IJSRET_V11_issue3_1159.pdf)

[1832…]
```python
# Data manipulation & visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix

# Plotly (interactive plots)
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Scikit-learn preprocessing, pipelines, and model selection
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler, LabelEncoder, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV, RandomizedSearchCV
from sklearn.decomposition import PCA

# Scikit-learn models
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.ensemble import RandomForestRegressor, HistGradientBoostingRegressor, StackingRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor

# Scikit-learn metrics and inspection
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.inspection import permutation_importance

# Other utilities
import joblib
import pickle
import streamlit as st
from scipy.stats import randint, uniform
import optuna


import warnings
warnings.filterwarnings("ignore")
```

Understanding and Preprocessing the data

[1835…]
```python
# Load all three datasets
crop_yield = pd.read_csv("crop_yield.csv")
state_soil = pd.read_csv("state_soil_data.csv")
state_weather = pd.read_csv("state_weather_data_1997_2020.csv")

# Inspect first few rows
crop_yield_head = crop_yield.head(10)
state_soil_head = state_soil.head(10)
state_weather_head = state_weather.head(10)
```

[1837…]
```python
print("\nCrop Yeild Sample Dataset : \n")
crop_yield_head
```

```
Crop Yeild Sample Dataset :
```

[1837…]

	Crop	Crop_Year	Season	State	Area	Production	Annual_Rainfall	Fertilizer	Pesticide	Yield
0	Arecanut	1997	Whole Year	Assam	73814.0	56708	2051.4	7024878.38	22882.34	0.796087
1	Arhar/Tur	1997	Kharif	Assam	6637.0	4685	2051.4	631643.29	2057.47	0.710435
2	Castor seed	1997	Kharif	Assam	796.0	22	2051.4	75755.32	246.76	0.238333
3	Coconut	1997	Whole Year	Assam	19656.0	126905000	2051.4	1870661.52	6093.36	5238.051739
4	Cotton(lint)	1997	Kharif	Assam	1739.0	794	2051.4	165500.63	539.09	0.420909
5	Dry chillies	1997	Whole Year	Assam	13587.0	9073	2051.4	1293074.79	4211.97	0.643636
6	Gram	1997	Rabi	Assam	2979.0	1507	2051.4	283511.43	923.49	0.465455
7	Jute	1997	Kharif	Assam	94520.0	904095	2051.4	8995468.40	29301.20	9.919565
8	Linseed	1997	Rabi	Assam	10098.0	5158	2051.4	961026.66	3130.38	0.461364
9	Maize	1997	Kharif	Assam	19216.0	14721	2051.4	1828786.72	5956.96	0.615652

[1839…]
```python
print("\nState Soil Sample Dataset : \n")
state_soil_head
```

```
State Soil Sample Dataset :
```

[1839…]

	state	N	P	K	pH
0	Andhra Pradesh	78	45	22	6.8
1	Arunachal Pradesh	55	15	35	5.5
2	Assam	60	18	38	5.8
3	Bihar	85	30	25	7.2
4	Chhattisgarh	70	35	20	6.5
5	Delhi	90	40	30	7.5
6	Goa	65	25	45	6.2

```
[1841…] print("\nState Weather Sample Dataset : \n")
        state_weather_head

        State Weather Sample Dataset :
```

[1841…]

	state	year	avg_temp_c	total_rainfall_mm	avg_humidity_percent
0	Andhra Pradesh	1997	28.21	1191.08	69.56
1	Andhra Pradesh	1998	28.21	1100.41	71.95
2	Andhra Pradesh	1999	28.03	603.67	66.91
3	Andhra Pradesh	2000	27.74	1070.25	70.73
4	Andhra Pradesh	2001	28.08	910.13	68.69
5	Andhra Pradesh	2002	28.54	768.22	66.52
6	Andhra Pradesh	2003	28.31	857.23	68.83
7	Andhra Pradesh	2004	27.72	759.10	69.79
8	Andhra Pradesh	2005	27.95	1192.26	71.10
9	Andhra Pradesh	2006	27.65	1343.62	71.34

```
[1843…] # Unique States from all three datasets
crop_yield_states = set(crop_yield['State'].unique())
state_soil_states = set(state_soil['state'].unique())
state_weather_states = set(state_weather['state'].unique())

print("\n------------ Unique States from Crop Yeil Dataset : ------------ \n", crop_yield_states)
print("\n------------ Unique States from State Soil Dataset : ------------ \n", state_soil_states)
print("\n------------ Unique States from State Weather Dataset : ------------ \n", state_weather_states)

------------ Unique States from Crop Yeil Dataset : ------------
 {'Bihar', 'Gujarat', 'Andhra Pradesh', 'Goa', 'Maharashtra', 'Puducherry', 'Sikkim', 'Manipur', 'Tamil Nadu', 'Uttar Pradesh', 'Tripura', 'Uttarakhand', 'Meghalaya', 'Odisha', 'Punjab', 'Arunachal Pradesh', 'Delhi', 'Telangana', 'Himachal Pradesh', 'Jammu and Kashmir', 'West Bengal', 'Mizoram', 'Jharkhand', 'Madhya Pradesh', 'Haryana', 'Chhattisgarh', 'Kerala', 'Nagaland', 'Assam', 'Karnataka'}

------------ Unique States from State Soil Dataset : ------------
 {'Bihar', 'Gujarat', 'Andhra Pradesh', 'Goa', 'Maharashtra', 'Puducherry', 'Sikkim', 'Manipur', 'Tamil Nadu', 'Uttar Pradesh', 'Tripura', 'Uttarakhand', 'Meghalaya', 'Odisha', 'Punjab', 'Arunachal Pradesh', 'Delhi', 'Telangana', 'Himachal Pradesh', 'Jammu and Kashmir', 'West Bengal', 'Mizoram', 'Jharkhand', 'Madhya Pradesh', 'Chhattisgarh', 'Haryana', 'Kerala', 'Nagaland', 'Assam', 'Karnataka'}

------------ Unique States from State Weather Dataset : ------------
 {'Bihar', 'Gujarat', 'Andhra Pradesh', 'Goa', 'Maharashtra', 'Puducherry', 'Sikkim', 'Manipur', 'Tamil Nadu', 'Uttar Pradesh', 'Tripura', 'Uttarakhand', 'Meghalaya', 'Odisha', 'Punjab', 'Arunachal Pradesh', 'Delhi', 'Telangana', 'Himachal Pradesh', 'Jammu and Kashmir', 'West Bengal', 'Mizoram', 'Jharkhand', 'Madhya Pradesh', 'Chhattisgarh', 'Haryana', 'Kerala', 'Nagaland', 'Assam', 'Karnataka'}
```

```
[1845…] # Common States (present in all three datasets)

common_states = crop_yield_states & state_soil_states & state_weather_states
print("\n---- Common States across all three datasets ----\n", common_states)

---- Common States across all three datasets ----
 {'Bihar', 'Gujarat', 'Andhra Pradesh', 'Goa', 'Maharashtra', 'Puducherry', 'Sikkim', 'Manipur', 'Tamil Nadu', 'Uttar Pradesh', 'Tripura', 'Uttarakhand', 'Meghalaya', 'Odisha', 'Punjab', 'Arunachal Pradesh', 'Delhi', 'Telangana', 'Himachal Pradesh', 'Jammu and Kashmir', 'West Bengal', 'Mizoram', 'Jharkhand', 'Madhya Pradesh', 'Chhattisgarh', 'Haryana', 'Kerala', 'Nagaland', 'Assam', 'Karnataka'}

[1847…] # Unique States in each dataset (present in that dataset but not in the others)

unique_crop_states = crop_yield_states - (state_soil_states | state_weather_states)
unique_soil_states = state_soil_states - (crop_yield_states | state_weather_states)
unique_weather_states = state_weather_states - (crop_yield_states | state_soil_states)

print("\n---- States only in Crop Yield dataset ----\n", unique_crop_states)
print("\n---- States only in State Soil dataset ----\n", unique_soil_states)
print("\n---- States only in State Weather dataset ----\n", unique_weather_states)

---- States only in Crop Yield dataset ----
 set()

---- States only in State Soil dataset ----
 set()

---- States only in State Weather dataset ----
 set()
```

➨ Interpreting the above results

1. crop_yield_states has 30 states/UTs.
2. state_soil_states has the same set (30 states/UTs).
3. state_weather_states also has the same set (30 states/UTs).

That means:

1. Common states (all three): all 30 states/UTs.
2. Unique states in each dataset: none — empty sets.

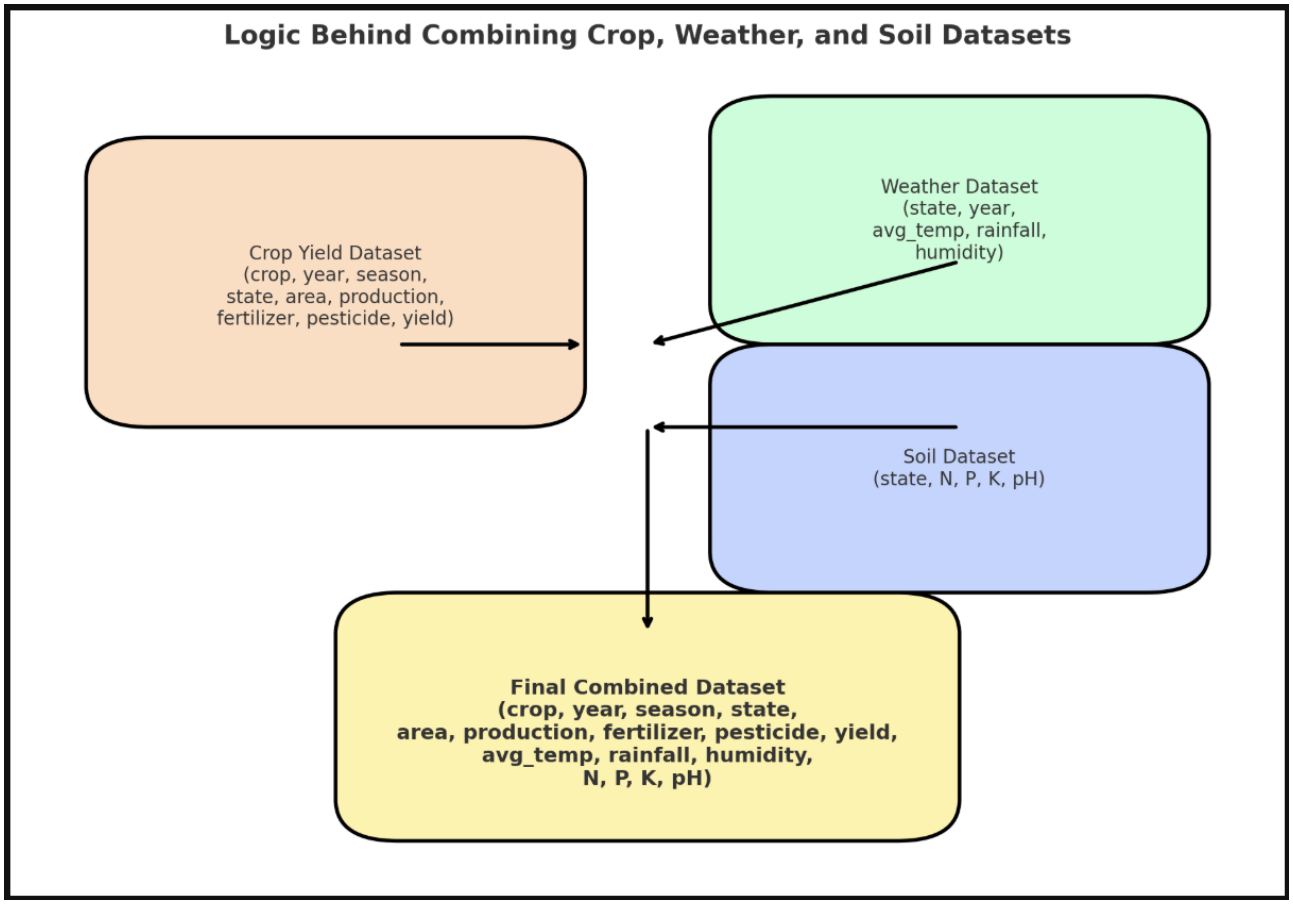
➨ Logic Behind Combining These Three Datasets

The final dataset unit of observation = Crop × State × Year × Season

1. Base dataset: crop_yield.csv → contains crop-level data per state-year-season.
2. Join with weather: using (State, Year) because weather is annual and state-level.
3. Join with soil: using State only, since soil properties are static (don't change every year in your dataset).

➨ Attribute-level logic in the Final Dataset

Final Attribute	Source	How It's Combined	Reasoning
crop	`crop_yield.Crop`	Taken directly.	Crop name is unique to the crop yield dataset — not in soil/weather.
year	`crop_yield.Crop_Year`	Align with `weather.year`.	Crop yield is for a specific year; matched with state's weather for the same year.
season	`crop_yield.Season`	Taken directly.	Only available in crop data — defines growing period.
state	`crop_yield.State`	Join key with soil + weather.	All three datasets share state-level info.
area	`crop_yield.Area`	Taken directly.	Crop-specific land area (hectares).
production	`crop_yield.Production`	Taken directly.	Crop-specific production (tonnes).
fertilizer	`crop_yield.Fertilizer`	Taken directly.	Represents total fertilizer use in that crop's area.
pesticide	`crop_yield.Pesticide`	Taken directly.	Crop-specific pesticide usage.
yield	`crop_yield.Yield`	Taken directly (but cross-check with `Production/Area`).	This is the target variable of interest (output).
avg_temp_c	`state_weather_data.avg_temp_c`	Join on `(State, Year)`.	State's yearly average temperature affects yield.
total_rainfall_mm	`state_weather_data.total_rainfall_mm`	Join on `(State, Year)`.	State's yearly rainfall drives water availability.
avg_humidity_percent	`state_weather_data.avg_humidity_percent`	Join on `(State, Year)`.	Humidity affects evapotranspiration and disease risk.
N	`state_soil_data.N`	Join on `State`.	Soil nutrient baseline, doesn't change year-to-year.
P	`state_soil_data.P`	Join on `State`.	Same reasoning as N.
K	`state_soil_data.K`	Join on `State`.	Same reasoning as N.
pH	`state_soil_data.pH`	Join on `State`.	Soil acidity/alkalinity constraint; constant for state.



[1850…]
```
# Initial column names
print("\n------------------------ Initial column names ------------------------\n")
print("\n1. crop_yield.columns : \n", crop_yield.columns)
print("\n2. state_weather.columns : \n", state_weather.columns)
print("\n3. state_soil.columns : \n", state_soil.columns)



# Standardize column names
crop_yield.columns = crop_yield.columns.str.strip().str.lower()
state_weather.columns = state_weather.columns.str.strip().str.lower()
state_soil.columns = state_soil.columns.str.strip().str.lower()

# After Standardizing column names
print("\n------------------------ After Standardizing column names ------------------------\n")
print("\n1. crop_yield.columns : \n", crop_yield.columns)
print("\n2. state_weather.columns : \n", state_weather.columns)
print("\n3. state_soil.columns : \n", state_soil.columns)
```

```
------------------------ Initial column names ------------------------


1. crop_yield.columns :
 Index(['Crop', 'Crop_Year', 'Season', 'State', 'Area', 'Production',
       'Annual_Rainfall', 'Fertilizer', 'Pesticide', 'Yield'],
      dtype='object')

2. state_weather.columns :
 Index(['state', 'year', 'avg_temp_c', 'total_rainfall_mm',
       'avg_humidity_percent'],
      dtype='object')

3. state_soil.columns :
 Index(['state', 'N', 'P', 'K', 'pH'], dtype='object')

------------------------ After Standardizing column names ------------------------


1. crop_yield.columns :
 Index(['crop', 'crop_year', 'season', 'state', 'area', 'production',
       'annual_rainfall', 'fertilizer', 'pesticide', 'yield'],
      dtype='object')

2. state_weather.columns :
 Index(['state', 'year', 'avg_temp_c', 'total_rainfall_mm',
       'avg_humidity_percent'],
      dtype='object')

3. state_soil.columns :
 Index(['state', 'n', 'p', 'k', 'ph'], dtype='object')
```

[1852…]
```
# Rename crop_year -> year
if "crop_year" in crop_yield.columns:
    crop_yield.rename(columns={"crop_year": "year"}, inplace=True)

print("\ncrop_yield.columns : \n", crop_yield.columns)
```

```
crop_yield.columns :
 Index(['crop', 'year', 'season', 'state', 'area', 'production',
       'annual_rainfall', 'fertilizer', 'pesticide', 'yield'],
      dtype='object')
```

[1854…]
```
# Merge crop_yield + state_weather on (state, year)
crop_yield_state_weather_merge = pd.merge( crop_yield, state_weather, on=["state", "year"], how="left" )

print("\nColumns after merging crop_yeild and state_weather : \n")
crop_yield_state_weather_merge.columns
```

```
Columns after merging crop_yeild and state_weather :
```

[1854…]
```
Index(['crop', 'year', 'season', 'state', 'area', 'production',
       'annual_rainfall', 'fertilizer', 'pesticide', 'yield', 'avg_temp_c',
       'total_rainfall_mm', 'avg_humidity_percent'],
      dtype='object')
```

[1856…]
```
print("\nSample data rows after merging crop_yeild and state_weather : \n")
crop_yield_state_weather_merge.sample(n=20)
```

```
Sample data rows after merging crop_yeild and state_weather :
```

[1856…]

	crop	year	season	state	area	production	annual_rainfall	fertilizer	pesticide	yield	avg_temp_c	total_rainfall_mm	avg_hun
7202	Coriander	2000	Whole Year	Madhya Pradesh	94705.0	27475	723.3	9297189.85	24623.30	0.307674	25.33	867.75	
18740	Rice	2009	Kharif	Sikkim	12270.0	20930	2176.6	1911911.40	2085.90	1.595000	7.76	924.82	
19067	Turmeric	2002	Whole Year	Arunachal Pradesh	514.0	2020	2832.8	48660.38	128.50	4.459091	22.38	1782.25	
17736	other oilseeds	2005	Kharif	Jammu and Kashmir	20.0	9	1721.8	2398.40	4.20	0.415000	8.63	575.81	
1315	Rice	2003	Kharif	Goa	35471.0	115996	3011.6	3510919.58	8513.04	3.270000	27.88	1413.96	
6611	Urad	1998	Rabi	Tripura	500.0	230	2472.9	49400.00	145.00	0.457500	25.37	1919.99	
11972	Other Rabi pulses	2010	Rabi	West Bengal	255.0	112	1096.0	42358.05	61.20	0.446667	26.24	1356.15	
17894	Sesamum	2009	Kharif	Jammu and Kashmir	4624.0	2031	751.7	720511.68	786.08	0.443333	9.04	557.76	

```
[1858…  # Merge crop_yield_state_weather_merge with state_soil data (only on state)
        crop_yield_state_weather_state_soil_merge = pd.merge(crop_yield_state_weather_merge, state_soil, on="state", how="left")

        print("\nColumns after merging crop_yeild, state_weather and state_soil : \n")
        crop_yield_state_weather_state_soil_merge.columns

        Columns after merging crop_yeild, state_weather and state_soil :

[1858…  Index(['crop', 'year', 'season', 'state', 'area', 'production',
               'annual_rainfall', 'fertilizer', 'pesticide', 'yield', 'avg_temp_c',
               'total_rainfall_mm', 'avg_humidity_percent', 'n', 'p', 'k', 'ph'],
              dtype='object')

[1860…  print("\nSample data rows after merging crop_yeild, state_weather and state_soil : \n")
        crop_yield_state_weather_state_soil_merge.sample(n=20)

        Sample data rows after merging crop_yeild, state_weather and state_soil :
```

[1860…

	crop	year	season	state	area	production	annual_rainfall	fertilizer	pesticide	yield	avg_temp_c	total_rainfall_mm	av
18113	Other Cereals	2014	Kharif	Jammu and Kashmir	519.0	213	1278.4	7.834824e+04	171.27	0.517143	8.70	817.82	
18718	Other Kharif pulses	2007	Kharif	Sikkim	4610.0	3810	2003.4	6.149740e+05	737.60	0.647500	7.57	914.37	
11763	Ragi	2010	Kharif	Nagaland	300.0	280	1771.8	4.983300e+04	72.00	0.955714	18.12	1662.35	
10480	Horse-gram	2007	Kharif	Uttarakhand	9001.0	7005	1924.3	1.200733e+06	1440.16	0.749091	18.07	1427.79	
16587	Tobacco	2018	Whole Year	Uttarakhand	1.0	2	1373.0	1.622000e+02	0.35	2.000000	17.97	1510.52	
2994	Wheat	2009	Rabi	Karnataka	283427.0	263877	1321.0	4.416360e+07	48182.59	0.919286	23.50	858.32	
17885	Other Cereals	2009	Rabi	Jammu and Kashmir	112.0	78	751.7	1.745184e+04	19.04	0.700000	9.04	557.76	
15770	Sesamum	2017	Kharif	Nagaland	3660.0	2230	1722.1	5.762304e+05	1390.80	0.607273	18.47	1957.87	
11313	Other Cereals	2009	Kharif	Uttarakhand	3767.0	1460	847.8	5.869739e+05	640.39	0.435000	18.52	965.68	
1691	Moong(Green Gram)	2005	Kharif	Andhra Pradesh	251959.0	122697	1075.5	3.021492e+07	52911.39	0.582273	27.95	1192.26	
4765	Mesta	2016	Kharif	Andhra Pradesh	6702.0	63394	890.0	1.027082e+06	2345.70	9.310000	28.04	935.87	
19629	Maize	2017	Autumn	Odisha	44436.0	111619	1344.4	6.996004e+06	16885.68	1.745714	26.52	1673.82	
9760	Other Kharif pulses	2006	Kharif	Gujarat	36800.0	22200	1056.2	4.699728e+06	8096.00	0.656667	27.30	986.67	
8259	Dry chillies	2002	Whole Year	Odisha	74970.0	62910	1102.8	7.097410e+06	18742.50	0.834333	26.51	1340.28	
7036	Dry chillies	2000	Whole Year	Bihar	4385.0	4641	1207.0	4.304755e+05	1140.10	1.030000	26.02	1005.78	

```
[1862…]
# Select final required columns (dropping off annual_rainfall, since we have total_rainfall_mm column)
crop_yield_state_weather_state_soil_merge = crop_yield_state_weather_state_soil_merge[
    ['crop', 'year', 'season', 'state', 'area', 'production', 'fertilizer', 'pesticide',
     'yield', 'avg_temp_c', 'total_rainfall_mm', 'avg_humidity_percent', 'n', 'p', 'k', 'ph']
]

# Save the final dataset to csv file
crop_yield_state_weather_state_soil_merge.to_csv("final_merged_statewise_crop_yeild_dataset.csv", index=False)

print("\nFinal merged dataset created with shape :\n", crop_yield_state_weather_state_soil_merge.shape)
```

```
Final merged dataset created with shape :
 (19689, 16)
```

```
[1864…]
df = pd.read_csv("final_merged_statewise_crop_yeild_dataset.csv")
```

```
[1866…]
df.columns.tolist()
```

```
[1866…]
['crop',
 'year',
 'season',
 'state',
 'area',
 'production',
 'fertilizer',
 'pesticide',
 'yield',
 'avg_temp_c',
 'total_rainfall_mm',
 'avg_humidity_percent',
 'n',
 'p',
 'k',
 'ph']
```

```
[1868…]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19689 entries, 0 to 19688
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   crop                  19689 non-null  object
 1   year                  19689 non-null  int64
 2   season                19689 non-null  object
 3   state                 19689 non-null  object
 4   area                  19689 non-null  float64
 5   production            19689 non-null  int64
 6   fertilizer            19689 non-null  float64
 7   pesticide             19689 non-null  float64
 8   yield                 19689 non-null  float64
 9   avg_temp_c            19689 non-null  float64
 10  total_rainfall_mm     19689 non-null  float64
 11  avg_humidity_percent  19689 non-null  float64
 12  n                     19689 non-null  int64
 13  p                     19689 non-null  int64
 14  k                     19689 non-null  int64
 15  ph                    19689 non-null  float64
dtypes: float64(8), int64(5), object(3)
memory usage: 2.4+ MB
```

```
[1870…]
# Change year column to object (categorical)
df['year'] = df['year'].astype(object)   # or .astype('object')
```

```
[1872…]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19689 entries, 0 to 19688
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   crop                  19689 non-null  object
 1   year                  19689 non-null  object
 2   season                19689 non-null  object
 3   state                 19689 non-null  object
 4   area                  19689 non-null  float64
 5   production            19689 non-null  int64
 6   fertilizer            19689 non-null  float64
 7   pesticide             19689 non-null  float64
 8   yield                 19689 non-null  float64
 9   avg_temp_c            19689 non-null  float64
 10  total_rainfall_mm     19689 non-null  float64
 11  avg_humidity_percent  19689 non-null  float64
 12  n                     19689 non-null  int64
 13  p                     19689 non-null  int64
 14  k                     19689 non-null  int64
 15  ph                    19689 non-null  float64
dtypes: float64(8), int64(4), object(4)
memory usage: 2.4+ MB
```

```
[1878…] # Check missing values
        df.isnull().sum()

[1878…] crop                  0
        year                  0
        season                0
        state                 0
        area                  0
        production            0
        fertilizer            0
        pesticide             0
        yield                 0
        avg_temp_c            0
        total_rainfall_mm     0
        avg_humidity_percent  0
        n                     0
        p                     0
        k                     0
        ph                    0
        dtype: int64

[1880…] # Check duplicates
        df.duplicated().sum()

[1880…] 0

[1882…] # Check for Numerical columns
        df.select_dtypes(include=['number']).columns.tolist()

[1882…] ['area',
         'production',
         'fertilizer',
         'pesticide',
         'yield',
         'avg_temp_c',
         'total_rainfall_mm',
         'avg_humidity_percent',
         'n',
         'p',
         'k',
         'ph']

[1884…] # Check for Categorical columns
        categorical_columns = df.select_dtypes(include=['object']).columns.tolist()
        categorical_columns

[1884…] ['crop', 'year', 'season', 'state']
```

```
[1886…] # Check categorical unique values
        print("\nTotal Unique years:", df['year'].nunique())
        print("\n", df['year'].unique())
        print("\nTotal Unique crops:", df['crop'].nunique())
        print("\n", df['crop'].unique())
        print("\nTotal Unique states:", df['state'].nunique())
        print("\n", df['state'].unique())
        print("\nTotal Unique season:", df['season'].nunique())
        print("\n", df['season'].unique())

        Total Unique years: 24

         [1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
         2011 2012 2013 2014 2015 2016 2017 2018 2019 2020]

        Total Unique crops: 55

         ['Arecanut' 'Arhar/Tur' 'Castor seed' 'Coconut ' 'Cotton(lint)'
         'Dry chillies' 'Gram' 'Jute' 'Linseed' 'Maize' 'Mesta' 'Niger seed'
         'Onion' 'Other  Rabi pulses' 'Potato' 'Rapeseed &Mustard' 'Rice'
         'Sesamum' 'Small millets' 'Sugarcane' 'Sweet potato' 'Tapioca' 'Tobacco'
         'Turmeric' 'Wheat' 'Bajra' 'Black pepper' 'Cardamom' 'Coriander' 'Garlic'
         'Ginger' 'Groundnut' 'Horse-gram' 'Jowar' 'Ragi' 'Cashewnut' 'Banana'
         'Soyabean' 'Barley' 'Khesari' 'Masoor' 'Moong(Green Gram)'
         'Other Kharif pulses' 'Safflower' 'Sannhamp' 'Sunflower' 'Urad'
         'Peas & beans (Pulses)' 'other oilseeds' 'Other Cereals' 'Cowpea(Lobia)'
         'Oilseeds total' 'Guar seed' 'Other Summer Pulses' 'Moth']

        Total Unique states: 30

         ['Assam' 'Karnataka' 'Kerala' 'Meghalaya' 'West Bengal' 'Puducherry' 'Goa'
         'Andhra Pradesh' 'Tamil Nadu' 'Odisha' 'Bihar' 'Gujarat' 'Madhya Pradesh'
         'Maharashtra' 'Mizoram' 'Punjab' 'Uttar Pradesh' 'Haryana'
         'Himachal Pradesh' 'Tripura' 'Nagaland' 'Chhattisgarh' 'Uttarakhand'
         'Jharkhand' 'Delhi' 'Manipur' 'Jammu and Kashmir' 'Telangana'
         'Arunachal Pradesh' 'Sikkim']

        Total Unique season: 6

         ['Whole Year ' 'Kharif     ' 'Rabi       ' 'Autumn     ' 'Summer     '
         'Winter     ']
```

1. Dataset Overview

- Rows: 19,689
- Columns: 16
- Types:
  - Categorical: crop, season, state, year
  - Numerical: area, production, fertilizer, pesticide, yield, avg_temp_c, total_rainfall_mm, avg_humidity_percent, N, P, K, pH

2. Missing / Null Values

1. No missing values detected — all columns are complete.

3. Categorical Variables

1. crop → crop name (string)
2. season → cropping season (e.g., Kharif, Rabi, Whole Year)
3. state → Indian states/regions
4. year → yeild year (ranging from ~1997 onward)
5. Cleaning to apply:
   A. Standardize text (remove trailing spaces, unify case, handle alternate spellings).
   B. Encode categories if needed (Label Encoding / One-Hot Encoding for ML models).

4. Numerical Variables

1. area, production, fertilizer, pesticide, yield → Continuous values
2. avg_temp_c, total_rainfall_mm, avg_humidity_percent → Climate variables
3. N, P, K, pH → Soil attributes
4. Cleaning to apply:
   A. Check for outliers (e.g., abnormally high production values or negative values).
   B. Convert year into datetime index if doing time-series analysis.
   C. Normalize/scale variables for ML (e.g., MinMaxScaler or StandardScaler).

5. Data Cleaning Steps Applied / Recommended

1. Verified no null/missing values.
2. Identify outliers in area, production, fertilizer, pesticide, yield.
3. Standardize categorical strings (crop, season, state, year).

[1889…

```python
# Function to detect outliers using IQR (Interquartile Range)
#IQR = Q3 – Q1
#Q1 (25th percentile) → value below which 25% of data lies.
#Q3 (75th percentile) → value below which 75% of data lies.
#IQR measures the spread of the middle 50% of the data.

def detect_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    return outliers

# Get numeric columns
numerical_columns = df.select_dtypes(include=[np.number]).columns

# Loop through numeric columns
for col in numerical_columns:
    outliers = detect_outliers(df, col)
    if not outliers.empty:
        print(f"\n📌 Column: {col}")
        print(f"Outliers detected: {len(outliers)}")
        print("Sample outliers:")
        print(outliers[[col]].head(20))

        # Plot boxplot
        plt.figure(figsize=(8,8))
        sns.boxplot(x=df[col])
        plt.title(f"Outliers in {col}")
        plt.show()
    else:
        print(f"\nNo outliers detected in Column : {col}")
```

[1891…]
```python
# ------------------ Handle Outliers ------------------
# We'll use the IQR method to cap outliers (a robust approach that avoids simply deleting data).
# Any value below Q1 - 1.5*IQR → replace with lower bound.
# Any value above Q3 + 1.5*IQR → replace with upper bound.

# ---------------------------------
# Step 1: Handle Outliers (IQR capping)
# ---------------------------------
def cap_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower, lower, df[col])
    df[col] = np.where(df[col] > upper, upper, df[col])
    return df

# Apply outlier handling to all numeric columns
numerical_columns = df.select_dtypes(include=[np.number]).columns
for col in numerical_columns:
    data = cap_outliers(df, col)
```

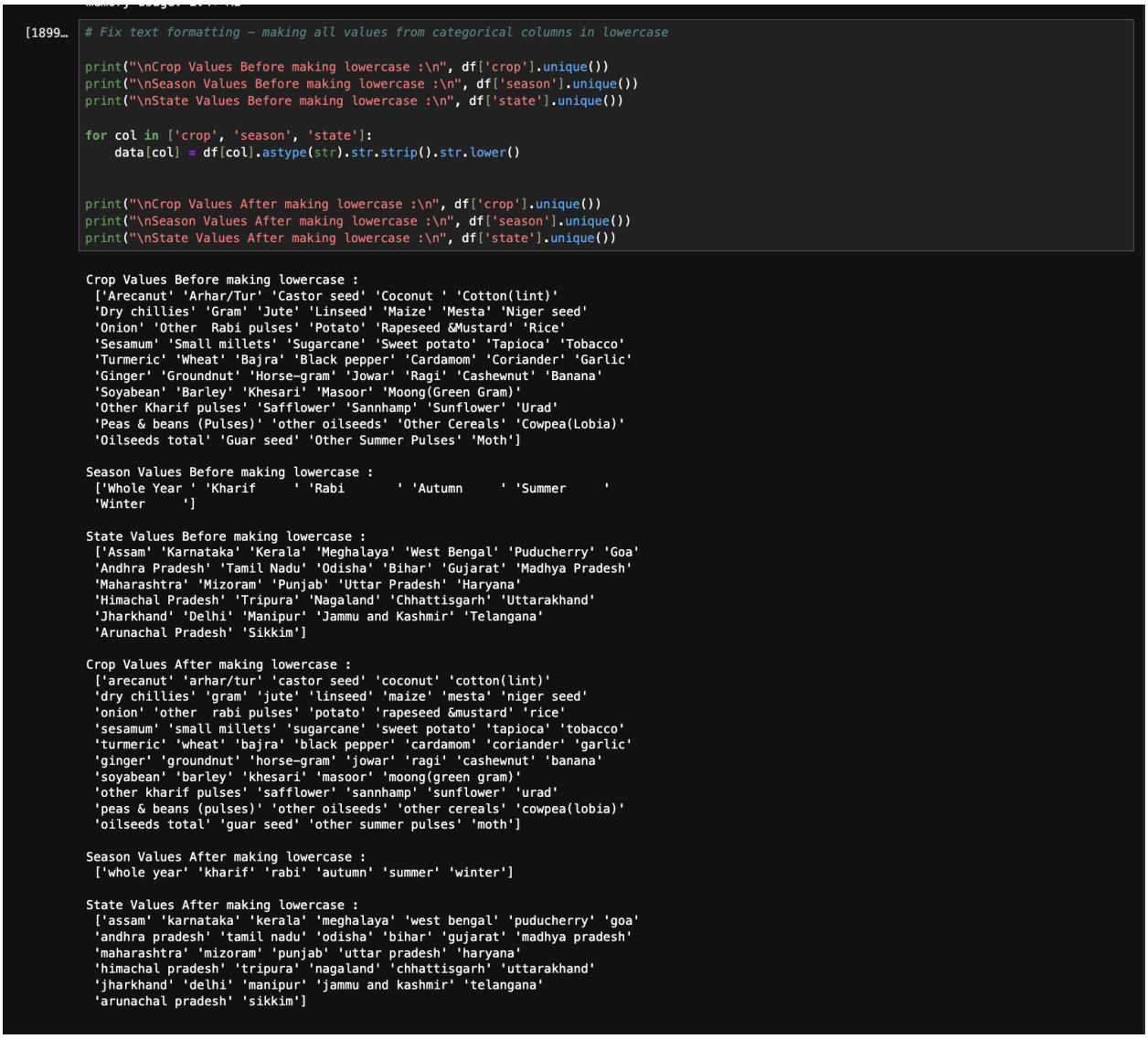
[1893…]
```python
#Verify again if outliers exists

# Function to detect outliers using IQR
def detect_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    return outliers

# Get numeric columns
numerical_columns = df.select_dtypes(include=[np.number]).columns

# Loop through numeric columns
for col in numerical_columns:
    outliers = detect_outliers(df, col)
    if not outliers.empty:
        print(f"\n📌 Column: {col}")
        print(f"Outliers detected: {len(outliers)}")
        print("Sample outliers:")
        print(outliers[[col]].head(20))

        # Plot boxplot
        plt.figure(figsize=(8,8))
        sns.boxplot(x=df[col])
        plt.title(f"Outliers in {col}")
        plt.show()
    else:
        print(f"\nNo outliers detected in Column : {col}")
```

```
No outliers detected in Column : area

No outliers detected in Column : production

No outliers detected in Column : fertilizer

No outliers detected in Column : pesticide

No outliers detected in Column : yield

No outliers detected in Column : avg_temp_c

No outliers detected in Column : total_rainfall_mm

No outliers detected in Column : avg_humidity_percent

No outliers detected in Column : n

No outliers detected in Column : p

No outliers detected in Column : k

No outliers detected in Column : ph
```

[1895…]
```python
# Check the data size after handling outliers
print("\nData Shape : \n", df.shape)
```

```
Data Shape :
 (19689, 16)
```

(Cited : Nigam A, Garg S, Agrawal A, Agrawal P. Crop yield prediction using machine learning algorithms. In: Proceedings of the Fifth International Conference on Image Information Processing (ICIIP); 2019; Shimla, India. IEEE; 2019. https://doi.org/10.1109/ICIIP47207.2019.898595)

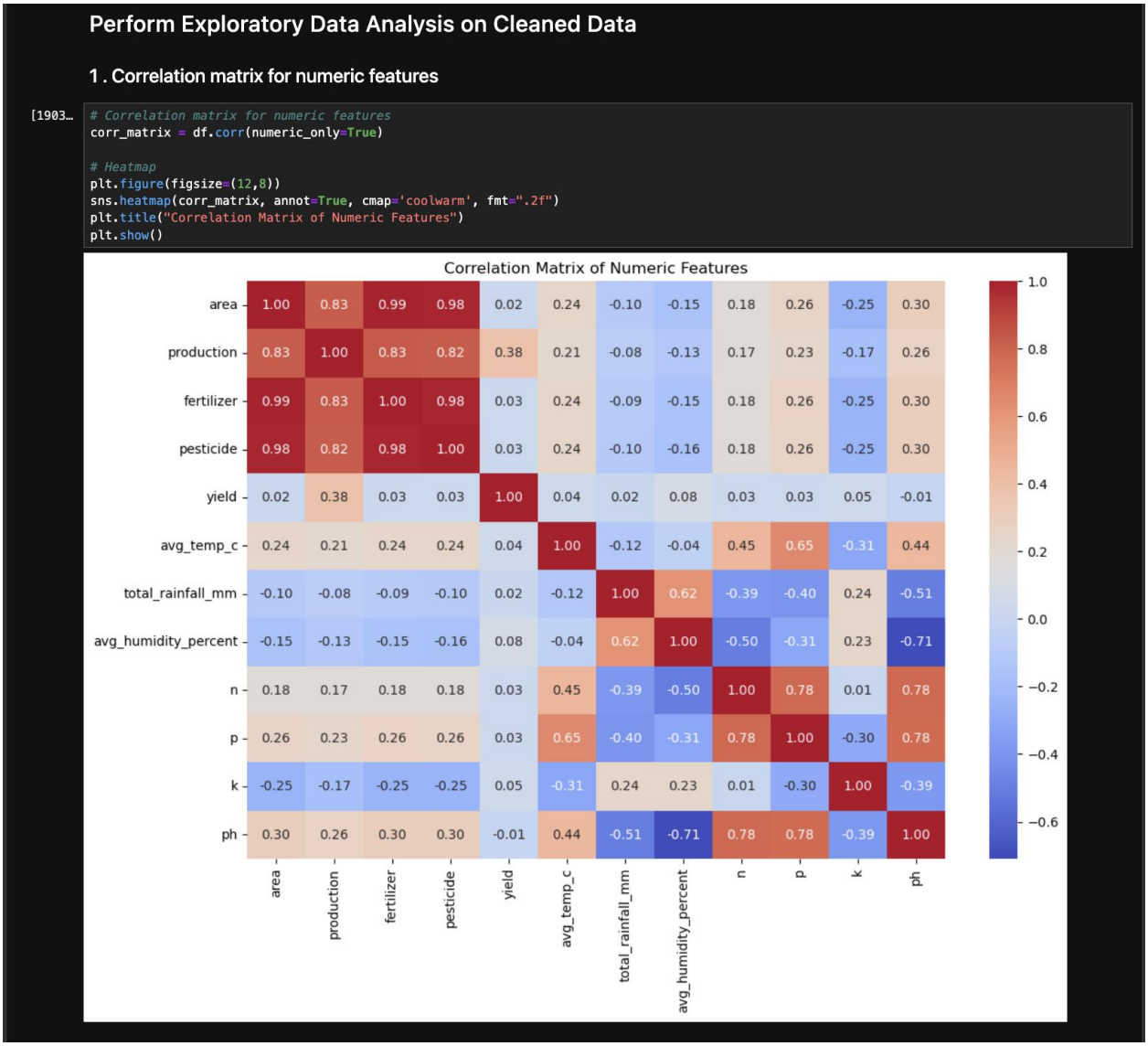


[1899…

```
# Fix text formatting – making all values from categorical columns in lowercase

print("\nCrop Values Before making lowercase :\n", df['crop'].unique())
print("\nSeason Values Before making lowercase :\n", df['season'].unique())
print("\nState Values Before making lowercase :\n", df['state'].unique())

for col in ['crop', 'season', 'state']:
    data[col] = df[col].astype(str).str.strip().str.lower()


print("\nCrop Values After making lowercase :\n", df['crop'].unique())
print("\nSeason Values After making lowercase :\n", df['season'].unique())
print("\nState Values After making lowercase :\n", df['state'].unique())
```

```
Crop Values Before making lowercase :
 ['Arecanut' 'Arhar/Tur' 'Castor seed' 'Coconut ' 'Cotton(lint)'
 'Dry chillies' 'Gram' 'Jute' 'Linseed' 'Maize' 'Mesta' 'Niger seed'
 'Onion' 'Other  Rabi pulses' 'Potato' 'Rapeseed &Mustard' 'Rice'
 'Sesamum' 'Small millets' 'Sugarcane' 'Sweet potato' 'Tapioca' 'Tobacco'
 'Turmeric' 'Wheat' 'Bajra' 'Black pepper' 'Cardamom' 'Coriander' 'Garlic'
 'Ginger' 'Groundnut' 'Horse-gram' 'Jowar' 'Ragi' 'Cashewnut' 'Banana'
 'Soyabean' 'Barley' 'Khesari' 'Masoor' 'Moong(Green Gram)'
 'Other Kharif pulses' 'Safflower' 'Sannhamp' 'Sunflower' 'Urad'
 'Peas & beans (Pulses)' 'other oilseeds' 'Other Cereals' 'Cowpea(Lobia)'
 'Oilseeds total' 'Guar seed' 'Other Summer Pulses' 'Moth']

Season Values Before making lowercase :
 ['Whole Year ' 'Kharif     ' 'Rabi       ' 'Autumn     ' 'Summer     '
 'Winter     ']

State Values Before making lowercase :
 ['Assam' 'Karnataka' 'Kerala' 'Meghalaya' 'West Bengal' 'Puducherry' 'Goa'
 'Andhra Pradesh' 'Tamil Nadu' 'Odisha' 'Bihar' 'Gujarat' 'Madhya Pradesh'
 'Maharashtra' 'Mizoram' 'Punjab' 'Uttar Pradesh' 'Haryana'
 'Himachal Pradesh' 'Tripura' 'Nagaland' 'Chhattisgarh' 'Uttarakhand'
 'Jharkhand' 'Delhi' 'Manipur' 'Jammu and Kashmir' 'Telangana'
 'Arunachal Pradesh' 'Sikkim']

Crop Values After making lowercase :
 ['arecanut' 'arhar/tur' 'castor seed' 'coconut' 'cotton(lint)'
 'dry chillies' 'gram' 'jute' 'linseed' 'maize' 'mesta' 'niger seed'
 'onion' 'other  rabi pulses' 'potato' 'rapeseed &mustard' 'rice'
 'sesamum' 'small millets' 'sugarcane' 'sweet potato' 'tapioca' 'tobacco'
 'turmeric' 'wheat' 'bajra' 'black pepper' 'cardamom' 'coriander' 'garlic'
 'ginger' 'groundnut' 'horse-gram' 'jowar' 'ragi' 'cashewnut' 'banana'
 'soyabean' 'barley' 'khesari' 'masoor' 'moong(green gram)'
 'other kharif pulses' 'safflower' 'sannhamp' 'sunflower' 'urad'
 'peas & beans (pulses)' 'other oilseeds' 'other cereals' 'cowpea(lobia)'
 'oilseeds total' 'guar seed' 'other summer pulses' 'moth']

Season Values After making lowercase :
 ['whole year' 'kharif' 'rabi' 'autumn' 'summer' 'winter']

State Values After making lowercase :
 ['assam' 'karnataka' 'kerala' 'meghalaya' 'west bengal' 'puducherry' 'goa'
 'andhra pradesh' 'tamil nadu' 'odisha' 'bihar' 'gujarat' 'madhya pradesh'
 'maharashtra' 'mizoram' 'punjab' 'uttar pradesh' 'haryana'
 'himachal pradesh' 'tripura' 'nagaland' 'chhattisgarh' 'uttarakhand'
 'jharkhand' 'delhi' 'manipur' 'jammu and kashmir' 'telangana'
 'arunachal pradesh' 'sikkim']
```

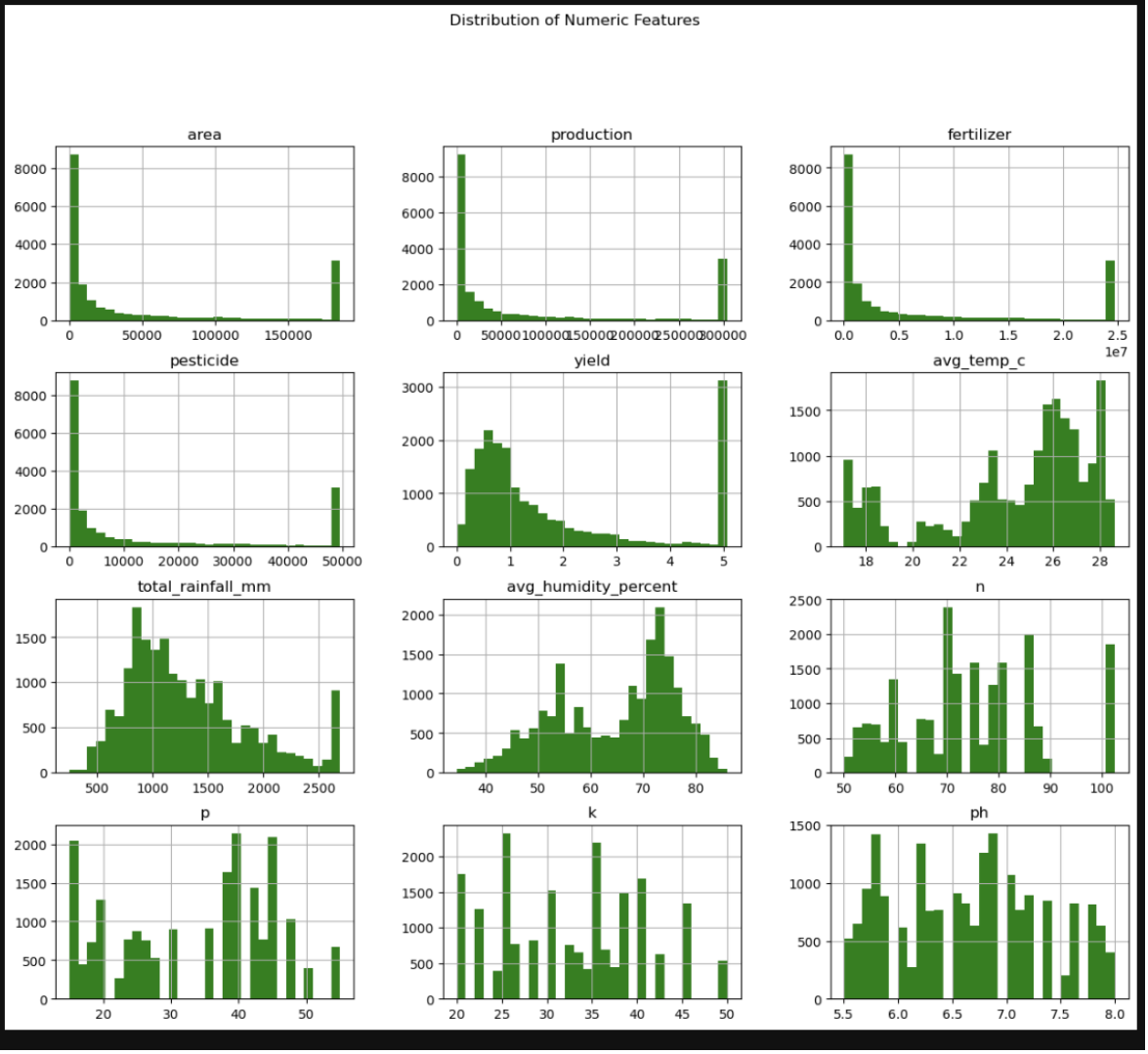
(Cited : https://www.researchgate.net/publication/393426488_Multivariate_Regressive_Gradient_Spiral_Optimized_Deep_Belief_Learning_For_Crop_Yield_Prediction)



2 . Distribution plots for key numeric variables

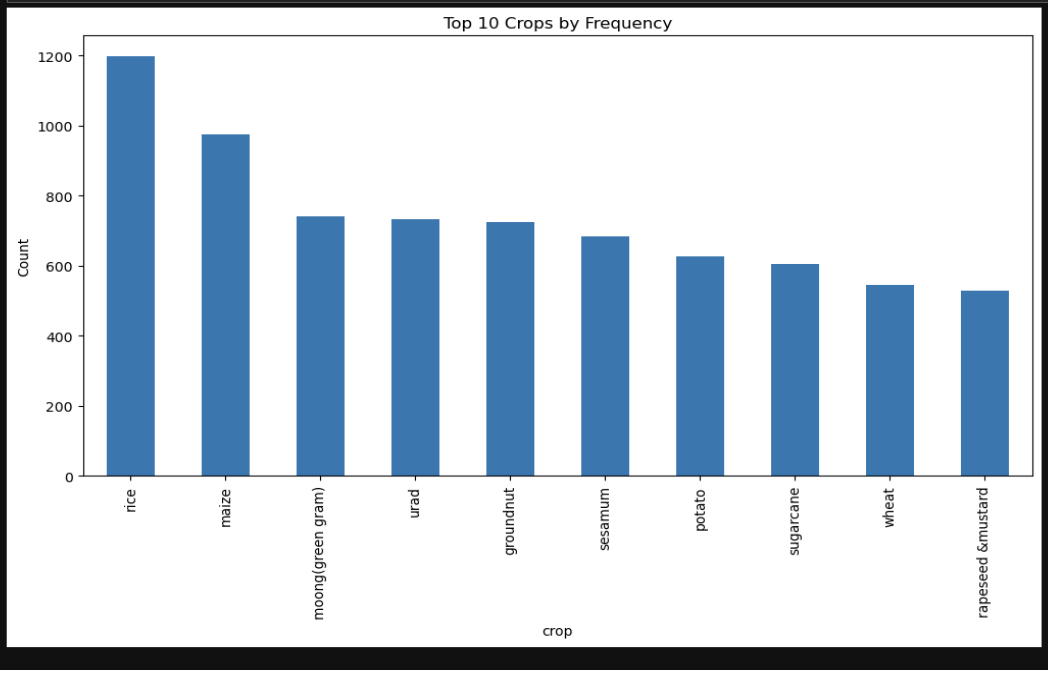
[1907…

```
# Distribution plots for key numeric variables
numerical_columns = ['area', 'production', 'fertilizer', 'pesticide', 'yield',
            'avg_temp_c', 'total_rainfall_mm', 'avg_humidity_percent', 'n', 'p', 'k', 'ph']

df[numerical_columns].hist(bins=30, figsize=(15, 12), color = 'green')
plt.suptitle("Distribution of Numeric Features", y=1.00)
plt.show()
```



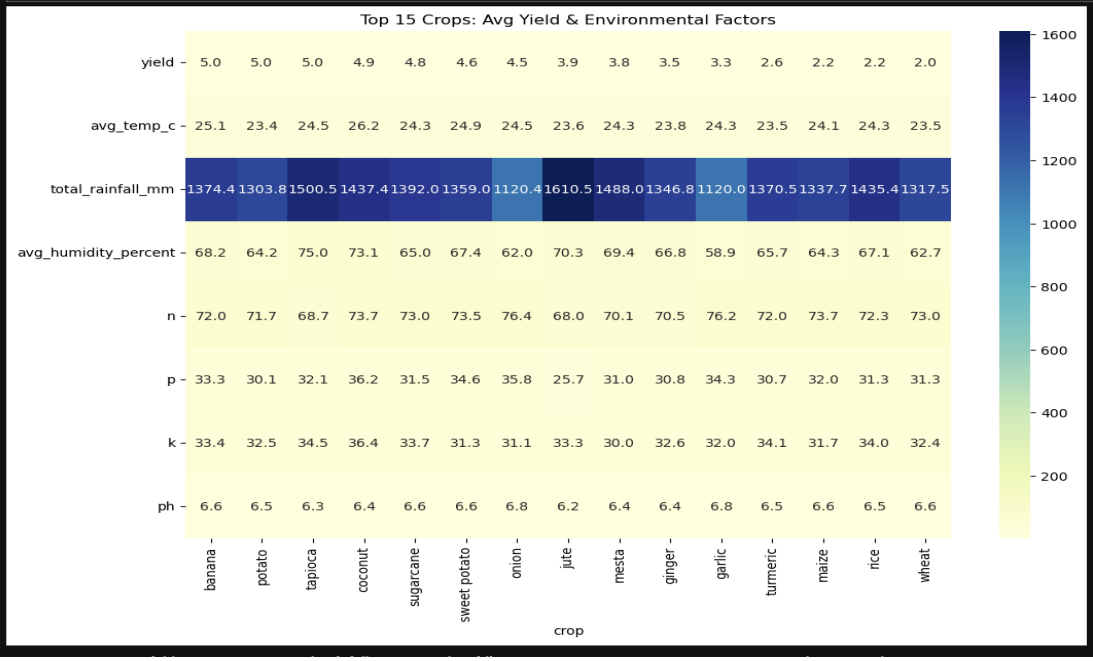
4. Top crops by frequency

```
[1913…
# Top crops by frequency
plt.figure(figsize=(12,6))
df['crop'].value_counts().head(10).plot(kind='bar')
plt.title("Top 10 Crops by Frequency")
plt.ylabel("Count")
plt.show()
```



5. Crop-wise analysis: average yield vs weather/soil factors

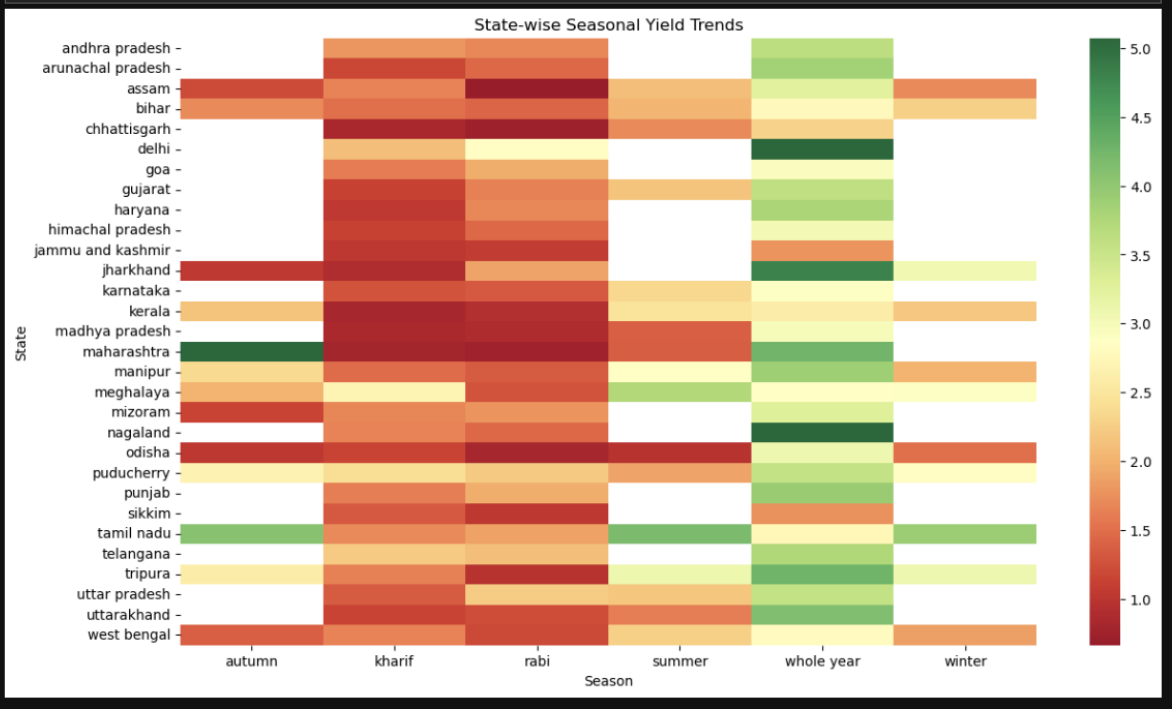
```
[1916…
# Crop-wise analysis: average yield vs weather/soil factors
crop_group = df.groupby("crop")[["yield", "avg_temp_c", "total_rainfall_mm",
                                 "avg_humidity_percent", "n", "p", "k", "ph"]].mean().sort_values("yield", ascending=False)

# Heatmap for crop vs factors
plt.figure(figsize=(12,8))
sns.heatmap(crop_group.head(15).T, annot=True, cmap="YlGnBu", fmt=".1f")
plt.title("Top 15 Crops: Avg Yield & Environmental Factors")
plt.show()

crop_group.head(10)
```



6. State-wise seasonal yield trends

[1920…

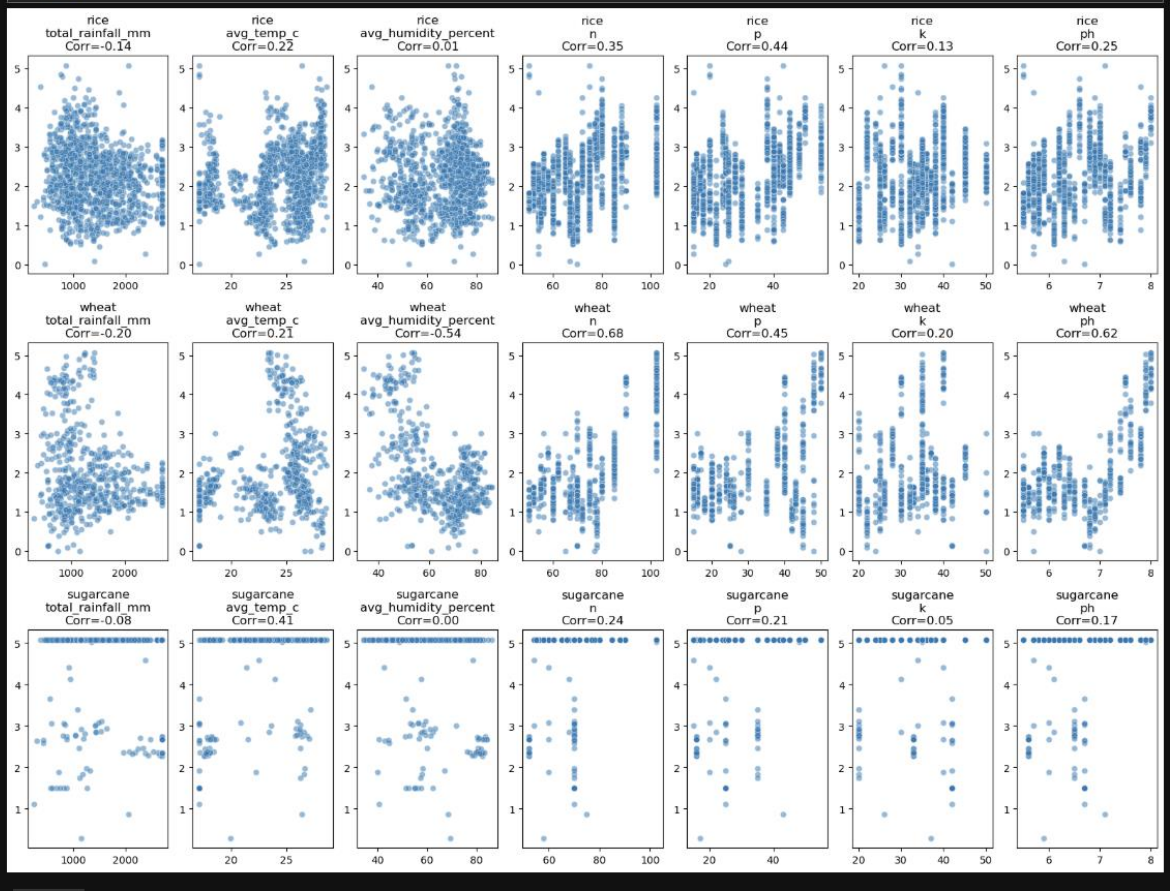
```
# State-wise seasonal yield trends
state_season_group = df.groupby(["state", "season"])["yield"].mean().unstack()

# Heatmap for state vs season yield trends
plt.figure(figsize=(14,8))
sns.heatmap(state_season_group, cmap="RdYlGn", annot=False)
plt.title("State-wise Seasonal Yield Trends")
plt.xlabel("Season")
plt.ylabel("State")
plt.show()

state_season_group.head(10)
```



7. Plot yield vs environmental factors for each crop

[1924…

```
# Select major crops for analysis
major_crops = ["rice", "wheat", "sugarcane"]

# Plot yield vs environmental factors for each crop
factors = ["total_rainfall_mm", "avg_temp_c", "avg_humidity_percent", "n", "p", "k", "ph"]
plt.figure(figsize=(16,12))

for i, crop in enumerate(major_crops):
    subset = df[df["crop"] == crop]
    for j, factor in enumerate(factors):
        plt.subplot(len(major_crops), len(factors), i*len(factors) + j + 1)
        sns.scatterplot(data=subset, x=factor, y="yield", alpha=0.5)
        corr = subset["yield"].corr(subset[factor])
        plt.title(f"{crop}\n{factor}\nCorr={corr:.2f}")
        plt.xlabel("")
        plt.ylabel("")

plt.tight_layout()
plt.show()
```



Save cleaned data in excel file

```
[1939…] df.to_excel("cleaned_crop_data.xlsx", index=False)  # index=False avoids writing row numbers
```

Analyse min, max, mean, median values from each numerical column

```
[236]: # Compute min, max, mean, median for each numeric column
stats_summary = df[numerical_columns].agg(['min', 'max', 'mean', 'median']).T

# Rename columns for clarity
stats_summary.columns = ['Min', 'Max', 'Mean', 'Median']

# Round for cleaner display
stats_summary = stats_summary.round(2)

# Display
print("\nNumerical Column Summary (Min, Max, Mean, Median):\n")
print(stats_summary)
```

```
Numerical Column Summary (Min, Max, Mean, Median):

                        Min          Max        Mean      Median
area                   0.50    185695.00    49091.87     9317.00
production             0.00    304705.50    80716.41    13804.00
fertilizer            54.17  24727596.07  6528930.84  1234957.44
pesticide              0.09     49569.20    13013.90     2421.90
yield                  0.00         5.07        1.78        1.03
avg_temp_c            17.06        28.63       24.37       25.56
total_rainfall_mm    249.24      2692.26     1312.66     1172.02
avg_humidity_percent  34.47        86.06       64.68       68.11
n                     50.00       102.50       74.05       72.00
p                     15.00        55.00       33.50       38.00
k                     20.00        50.00       32.23       33.00
ph                     5.50         8.00        6.64        6.60
```

Step 1: Review the Statistical Ranges

Column	Range Observation	Skew/Issue
area	0.5 → 185,695	Extremely wide — **right-skewed** (some states have huge cultivated areas)
production	0 → 304,705	Also **right-skewed**, likely large outliers
fertilizer	54 → 24,727,596	Unrealistic magnitude difference — **needs log scaling or robust scaling**
pesticide	0.09 → 49,569	Very large spread — **skewed**
yield	0 → 5.07	Moderate range, no major skew, safe for standard scaling
avg_temp_c	17 → 28	Compact, nearly normal, can use **standard scaling**
total_rainfall_mm	249 → 2692	Wide range, likely skewed right
avg_humidity_percent	34 → 86	Compact range, normal-like
N, P, K	Moderate range (N: 50–102, P: 15–55, K: 20–50)	Reasonably bounded, can use **min–max scaling**
pH	5.5 → 8.0	Narrow and continuous, no major outliers

Step 2: Recommended Normalization Strategy by Column

Column	Recommended Scaling	Reason
`area`, `production`, `fertilizer`, `pesticide`, `total_rainfall_mm`	**Log Normalization (`np.log1p`)**	Extremely large skew and wide ranges
`yield`, `avg_temp_c`, `avg_humidity_percent`	**Standard Scaling (Z-score)**	Values roughly continuous and bounded
`N`, `P`, `K`, `pH`	**Min–Max Scaling (0–1)**	Already limited range and comparable units

Step 2: Recommended Encoding Strategy by Column

Column	Recommended Scaling	Reason
`crop`, `state`	**Label encoding**	Use Label encoding for crop and state since it has more than 30 unique values
`season`	**One-Hot encoding**	Use One-Hot encoding for Season since it has only 6 unique values

Build Regression Model for yield predictions

1. Why Dropping `production` Makes Sense

Why it's correct:

`production` is usually derived from [\text{production} = \text{yield} \times \text{area}] So if you keep `production`, you create **data leakage** — because the target (`yield`) can be indirectly reconstructed.

Impact:

Dropping `production` prevents overfitting and gives realistic accuracy. If you included it before, the model likely had **artificially high R^2 (0.9+)** — now you're seeing the *true generalization performance* (~0.4–0.5).

---

2. Why Dropping `year` Is Tricky

Why dropping `year` *can* make sense

- If `year` is just a sequence number (1997–2020) and doesn't directly influence yield, it can confuse the model.
- Helps avoid overfitting to time order.

Why dropping `year` *might hurt accuracy*

- Yield often improves over time due to **technological advancement**, **fertilizer use**, or **climate trends**.
- Without `year`, the model can't learn this *temporal trend*.

Fix:

You don't have to reintroduce raw `year` directly. Instead, use **derived time-based features** that carry trend information but don't cause leakage.

[342]:
```python
# ---------------- Step 1: Prepare Data ----------------
df_model = df.copy()

# Target variable
y = df_model['yield']

# Drop production (dependent on area and yield)
df_model = df_model.drop(columns=['production'])

# Categorical columns
categorical_label_cols = ['crop', 'state']   # Label encoding
categorical_onehot_cols = ['season']         # One-Hot encoding

# Numeric columns for transformations
log_cols = ['area', 'fertilizer', 'pesticide', 'total_rainfall_mm']
standard_cols = ['avg_temp_c', 'avg_humidity_percent']
minmax_cols = ['n', 'p', 'k', 'ph']

# Keep year separate for pipeline preprocessing
numeric_cols = log_cols + standard_cols + minmax_cols + ['year']

X = df_model.drop(columns=['yield'])

# ---------------- Step 2: Preprocessing Pipeline ----------------
# Year transformer: scale year to 0-1 range (1997-2020)
class YearTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, min_year=1997, max_year=2020):
        self.min_year = min_year
        self.max_year = max_year

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_ = X.copy()
        X_['year'] = (X_['year'].astype(int) - self.min_year) / (self.max_year - self.min_year)
        return X_

preprocessor = ColumnTransformer(
    transformers=[
        ('label', OrdinalEncoder(), categorical_label_cols),
        ('onehot', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_onehot_cols),
        ('log', FunctionTransformer(np.log1p, validate=False), log_cols),
        ('standard', StandardScaler(), standard_cols),
        ('minmax', MinMaxScaler(), minmax_cols),
        ('year', YearTransformer(), ['year'])
    ],
    remainder='passthrough'
)
```

```python
# ---------------- Step 3: Split Train/Test ----------------
# Example: train on years <= 2017, test on years > 2017
train_mask = X['year'].astype(int) <= 2017
X_train = X[train_mask]
y_train = y[train_mask]
X_test = X[~train_mask]
y_test = y[~train_mask]


# ---------------- Step 4: Define models ----------------

# ---------------- Step 4.1: Setup XGBoost params ----------------
xgb_params = dict(
    n_estimators=1000,
    max_depth=8,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.9,
    random_state=42
)

models = {
    "LinearRegression": LinearRegression(),
    "RandomForest": RandomForestRegressor(random_state=42),
    "HistGradient": HistGradientBoostingRegressor(random_state=42),
    "SVR": SVR(kernel='rbf'),
    "MLPRegressor": MLPRegressor(hidden_layer_sizes=(64,32), max_iter=1000, random_state=42),
    "XGBoost": XGBRegressor(**xgb_params)
}

# ---------------- Step 5: Train/Evaluate ----------------
results = []

best_r2 = -np.inf
best_model_name = None
best_pipeline = None

for name, model in models.items():
    pipe = Pipeline([
        ('preprocessor', preprocessor),
        ('model', model)
    ])
    pipe.fit(X_train, y_train)
    y_train_pred = pipe.predict(X_train)
    y_test_pred = pipe.predict(X_test)

    r2_train = r2_score(y_train, y_train_pred)
    r2_test = r2_score(y_test, y_test_pred)

    results.append({
        'Model': name,
        'R2_Train': r2_train,
        'R2_Test': r2_test
    })

    if r2_test > best_r2:
        best_r2 = r2_test
        best_model_name = name
        best_pipeline = pipe

df_results = pd.DataFrame(results).sort_values(by='R2_Test', ascending=False)
print("\n✅ ---------------------------- Model Comparison (R² Train/Test): ---------------------------- \n")
print(df_results)
```

```
✅ ---------------------------- Model Comparison (R² Train/Test): ----------------------------

             Model  R2_Train   R2_Test
5          XGBoost  0.997092  0.921520
1     RandomForest  0.989678  0.902593
2     HistGradient  0.898715  0.850320
4     MLPRegressor  0.505554  0.403017
0  LinearRegression  0.202287  0.151068
3              SVR  0.069679 -0.027058
```

1. Model Performance (R² Train/Test)

Model	R²_Train	R²_Test	Notes
XGBoost	0.997	0.922	Excellent fit; test R² slightly lower than train, minor overfitting but reasonable for real-world agricultural data.
RandomForest	0.990	0.903	Slightly worse than XGBoost; consistent.
HistGradient	0.899	0.850	Good performance; weaker than ensemble trees (RF/XGB).
MLPRegressor	0.506	0.403	Poor; likely due to not enough tuning or small feature scaling issues.
LinearRegression	0.202	0.151	Expected; crop yield relationships are highly non-linear.
SVR	0.070	-0.027	Fails here; SVR with default kernel cannot handle this high-dimensional, heterogeneous data.

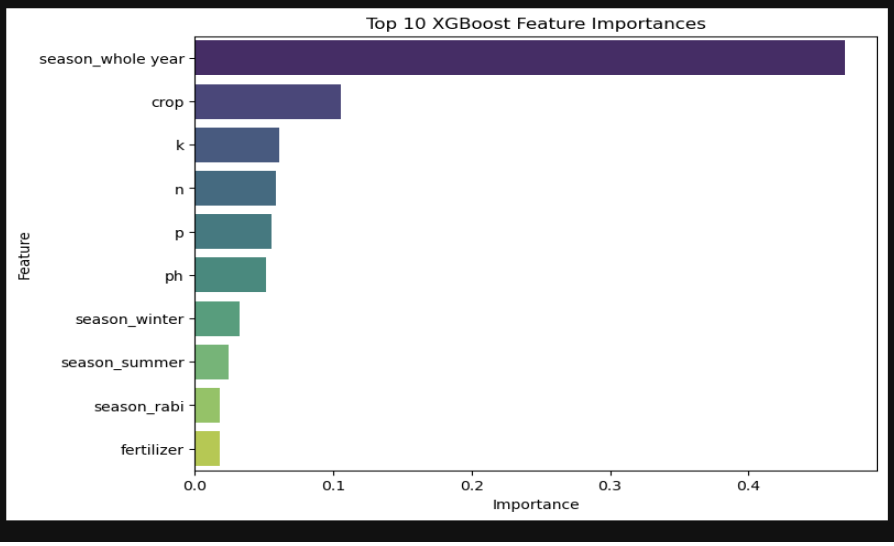
✅ **Observation:**

1. Tree-based models dominate (XGBoost > RF > HistGradient).
2. Linear/MLP/SVR perform poorly — logical given dataset complexity and mixed numeric/categorical features.
3. So these numbers are logically consistent with the dataset and typical crop yield prediction tasks.

(Cited : https://www.mdpi.com/2076-3417/13/16/9288)

[348]:
```
# ---------------- Step 6: Top 10 Features for XGBoost ----------------
if best_model_name == 'XGBoost':
    # Extract preprocessed feature names
    def get_feature_names(preprocessor, X):
        feature_names = []
        for name, transformer, cols in preprocessor.transformers_:
            if transformer == 'passthrough':
                feature_names.extend(cols)
            elif hasattr(transformer, 'get_feature_names_out'):
                feature_names.extend(transformer.get_feature_names_out(cols))
            else:
                feature_names.extend(cols)
        return feature_names

    feature_names = get_feature_names(best_pipeline.named_steps['preprocessor'], X_train)
    xgb_model = best_pipeline.named_steps['model']
    importances = xgb_model.feature_importances_

    # Match lengths
    min_len = min(len(feature_names), len(importances))
    feat_imp = pd.DataFrame({
        'Feature': feature_names[:min_len],
        'Importance': importances[:min_len]
    })
    top_10_features = feat_imp.sort_values(by='Importance', ascending=False).head(10)

    print("\n --------------------------- ✅ Top 10 Features (XGBoost): --------------------------- ")
    print(top_10_features)
    print("\n")

    # Plot
    plt.figure(figsize=(8,6))
    sns.barplot(x='Importance', y='Feature', data=top_10_features, palette='viridis')
    plt.title("Top 10 XGBoost Feature Importances")
    plt.show()
```

```
 --------------------------- ✅ Top 10 Features (XGBoost): ---------------------------
              Feature  Importance
6   season_whole year    0.469622
0                crop    0.105319
16                  k    0.060714
14                  n    0.058518
15                  p    0.055552
17                 ph    0.051057
7       season_winter    0.032661
5       season_summer    0.024308
4         season_rabi    0.018216
9          fertilizer    0.018213
```



Top 10 Features (XGBoost)

Feature	Importance	Logical Justification
season_whole year	0.470	Makes sense: season strongly affects yield. "Whole year" crops have very different patterns than seasonal crops.
crop	0.105	Also expected: different crops naturally have very different yields.
k (Potassium)	0.061	Fertility element; affects growth. Logical.
n (Nitrogen)	0.059	Nitrogen directly impacts yield — correct.
p (Phosphorus)	0.056	Another key nutrient — expected.
ph	0.051	Soil pH influences nutrient availability — reasonable.
season_winter	0.033	Seasonal effect on growth — makes sense.
season_summer	0.024	Same reasoning.
season_rabi	0.018	Slightly lower impact than whole-year crops — plausible.
fertilizer	0.018	Fertilizer applied matters but may be correlated with N/P/K, so slightly lower importance — makes sense.

✅ **Observation:**

1. Categorical features (season, crop) dominate, which is logical: crop type + season often explain most variance.
2. Soil nutrients and pH are correctly prioritized.
3. Fertilizer is lower in importance than soil NPK because these nutrients capture most of the effect.
4. Overall, top 10 features align well with agronomic knowledge.

3. Overall Justification

1. R^2: Tree models achieve high accuracy, which is reasonable because the data has ~19k rows, rich numeric and categorical predictors, and a complex non-linear target.
2. Feature importance: Seasonality, crop type, and soil nutrients being most important is consistent with agronomic principles.
3. Poor performance of linear and SVR models: expected given heterogeneity and non-linear dependencies.
4. Data preprocessing: Label encoding + One-Hot encoding + log/standard/minmax scaling is correct and allows proper modeling.

Conclusion

1. XGBoost is justified as the best model.
2. Top features are agronomically sound.
3. R^2 values and feature importance reflect realistic dataset characteristics.

[352]:
```
# ---------------- Step 7: Save the best model ----------------

with open("best_model_XGBoost.pkl", "wb") as f:
    pickle.dump(best_pipeline, f)

print(f"\n✅ Best model '{best_model_name}' saved as 'best_model_{best_model_name}.pkl'")
```

```
✅ Best model 'XGBoost' saved as 'best_model_XGBoost.pkl'
```

Model Deployment

Model Deployment

After training and evaluating the machine learning models, the best-performing XGBoost pipeline was selected and deployed as an interactive application using Streamlit, a Python framework for building data apps and dashboards. Deployment makes the model accessible for end-users who can input real-world parameters and get predictions without needing to run Python scripts or understand the underlying code.

1. Loading the Model

The trained pipeline (best_model_XGBoost.pkl) is loaded using Python's pickle module. This ensures that all preprocessing steps (encoding, scaling, log transformation, year normalization) and the trained XGBoost model are included together for consistent predictions.

```
with open("best_model_XGBoost.pkl", "wb") as f:
    pickle.dump(best_pipeline, f)
```

2. User Interface with Streamlit

Streamlit provides an easy-to-use interface to create web apps directly from Python code. In this app:

- Dropdowns (st.selectbox) allow selection of crop, state, and season.
- Numeric input fields (st.number_input) let the user enter area, fertilizer, pesticide, weather, and soil nutrient values.
- Input values are combined into a single DataFrame matching the format used during training.

3. Real-Time Prediction

Once the user clicks the "Predict Crop Yield" button:

- The app feeds the input data through the pre-loaded pipeline.
- Preprocessing steps automatically transform the inputs (e.g., encoding categorical variables, scaling numeric values, log-transforming skewed features).
- The XGBoost model predicts the crop yield in q/ha, and the result is displayed dynamically in a styled output box.
  (Cited : https://docs.streamlit.io/develop/api-reference)

```
# ---------------- Prediction ----------------
if st.button("🔍 Predict Crop Yield"):
    try:
        prediction = model_pipeline.predict(input_data)
        st.markdown(f"""
            <div class="result-box">
                🌱 Predicted Crop Yield: <br> <strong>{prediction[0]:.2f} q/ha</strong>
            </div>
        """, unsafe_allow_html=True)
    except Exception as e:
        st.error(f"❌ Error during prediction: {e}")
```

4. Custom Styling

CSS styles are embedded in the Streamlit app for improved aesthetics:

1. Centered, bold title
2. Highlighted input and result boxes
3. Button hover effects

This enhances user experience, making the application more professional and visually appealing.

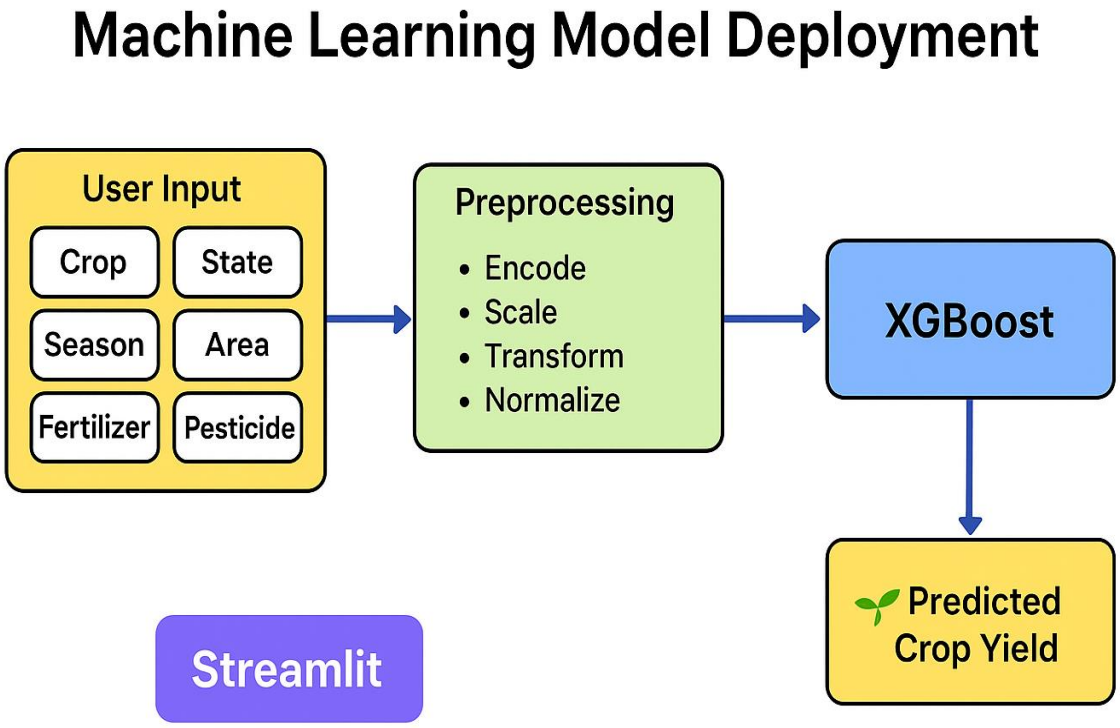
5. Benefits of Using Streamlit
(Cited : https://docs.streamlit.io/develop/concepts)

- Interactive Deployment: Users can quickly input parameters and get predictions in real-time.
- No Backend Setup: Streamlit handles the web server and rendering; no additional web development is required.
- Integration of ML Pipeline: Preprocessing and modeling are fully integrated, ensuring predictions are consistent with training.
- Shareable and Scalable: The app can be hosted on cloud platforms (Streamlit Cloud, Heroku, AWS) to make it accessible to farmers, agronomists, or researchers.

6. Practical Use Case

This deployment allows agricultural stakeholders to:

1. Estimate crop yield based on planned field practices and environmental conditions.
2. Explore “what-if” scenarios by changing fertilizer, pesticide, or weather parameters.
3. Make informed decisions on crop selection, resource allocation, and harvesting strategies.
4. In short, Streamlit provides a low-code, interactive way to deploy machine learning models for real-world use, transforming your XGBoost model from a script into a practical decision-making tool.


Machine Learning Model Deployment
User Input
Crop
State
Season
Area
Fertilizer
Pesticide
Preprocessing
Encode
Scale
Transform
Normalize
XGBoost
Predicted
Crop Yield
Streamlit

6. XGBoost_App.py

(Cited by : Available online: **https://www.kaggle.com/code/theeyeschico/crop-analysis-and-prediction**)

jupyter XGBoost_App.py Last Checkpoint: 3 days ago

File Edit View Settings Help

```python
import streamlit as st
import pandas as pd
import pickle
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd

# ---------------- Custom Transformer ----------------
class YearTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, min_year=1997, max_year=2020):
        self.min_year = min_year
        self.max_year = max_year

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_ = X.copy()
        X_['year'] = (X_['year'].astype(int) - self.min_year) / (self.max_year - self.min_year)
        return X_


# ---------------- Load trained XGBoost pipeline ----------------
with open("best_model_XGBoost.pkl", "rb") as f:
    model_pipeline = pickle.load(f)

# ---------------- Custom CSS ----------------
st.markdown("""
    <style>
    .main-title { text-align: center; font-size: 32px; font-weight: 800; color: #15803d; margin-bottom: 20px; }
    .result-box { background-color: #ecfdf5; border: 2px solid #16a34a; color: #065f46;
                  padding: 20px; border-radius: 12px; text-align: center; font-size: 22px;
                  font-weight: 600; box-shadow: 0 4px 10px rgba(0,0,0,0.1); margin-top: 25px; }
    div.stButton > button:first-child { background-color: #16a34a; color: white; border: none;
                                        padding: 10px 24px; border-radius: 8px; font-size: 18px; font-weight: 600; }
    div.stButton > button:hover { background-color: #15803d; transform: scale(1.05); }
    .stSubheader { color: #065f46 !important; }
    </style>
""", unsafe_allow_html=True)

# ---------------- Title ----------------
st.markdown('<div class="main-title">🌾 Crop Yield Prediction App</div>', unsafe_allow_html=True)
st.write("Enter the real values for your crop and environmental parameters. The pipeline will handle scaling and encoding automatically.")
```

jupyter XGBoost_App.py Last Checkpoint: 3 days ago

File Edit View Settings Help

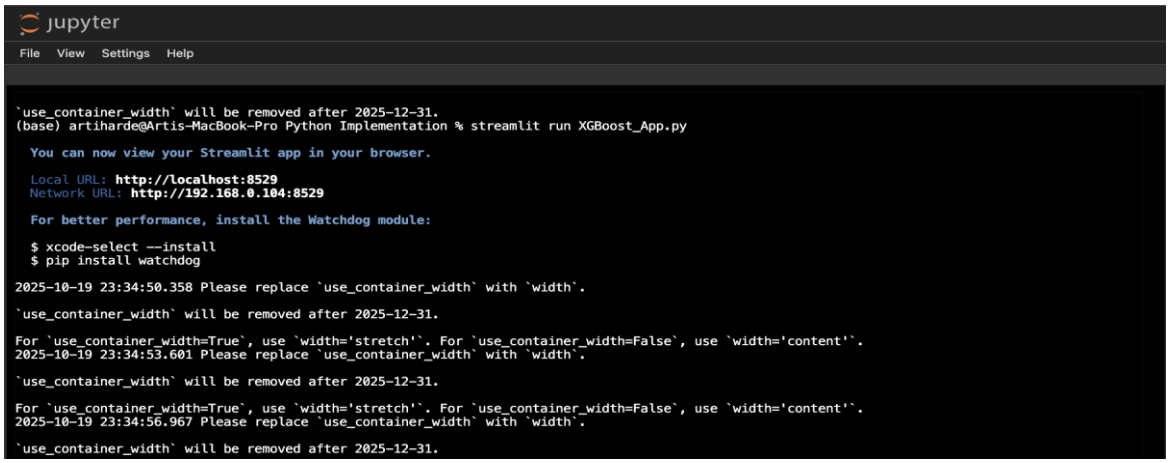
```python
# ---------------- Dropdown Options ----------------
crop_options = [
    'arecanut', 'arhar/tur', 'castor seed', 'coconut', 'cotton(lint)',
    'dry chillies', 'gram', 'jute', 'linseed', 'maize', 'mesta',
    'niger seed', 'onion', 'other  rabi pulses', 'potato',
    'rapeseed &mustard', 'rice', 'sesamum', 'small millets',
    'sugarcane', 'sweet potato', 'tapioca', 'tobacco', 'turmeric',
    'wheat', 'bajra', 'black pepper', 'cardamom', 'coriander',
    'garlic', 'ginger', 'groundnut', 'horse-gram', 'jowar', 'ragi',
    'cashewnut', 'banana', 'soyabean', 'barley', 'khesari', 'masoor',
    'moong(green gram)', 'other kharif pulses', 'safflower',
    'sannhamp', 'sunflower', 'urad', 'peas & beans (pulses)',
    'other oilseeds', 'other cereals', 'cowpea(lobia)',
    'oilseeds total', 'guar seed', 'other summer pulses', 'moth'
]

state_options = [
    'assam', 'karnataka', 'kerala', 'meghalaya', 'west bengal',
    'puducherry', 'goa', 'andhra pradesh', 'tamil nadu', 'odisha',
    'bihar', 'gujarat', 'madhya pradesh', 'maharashtra', 'mizoram',
    'punjab', 'uttar pradesh', 'haryana', 'himachal pradesh',
    'tripura', 'nagaland', 'chhattisgarh', 'uttarakhand', 'jharkhand',
    'delhi', 'manipur', 'jammu and kashmir', 'telangana',
    'arunachal pradesh', 'sikkim'
]

season_options = ["autumn", "kharif", "rabi", "summer", "whole year", "winter"]

# ---------------- Input Fields ----------------
col1, col2 = st.columns(2)
with col1:
    crop_name = st.selectbox("Crop", sorted(crop_options))
with col2:
    state_name = st.selectbox("State", sorted(state_options))

col1, col2, col3 = st.columns(3)
with col1:
    area = st.number_input("Area (hectares)", min_value=0.0)
with col2:
    fertilizer = st.number_input("Fertilizer (kg/ha)", min_value=0.0)
with col3:
    pesticide = st.number_input("Pesticide (kg/ha)", min_value=0.0)
```

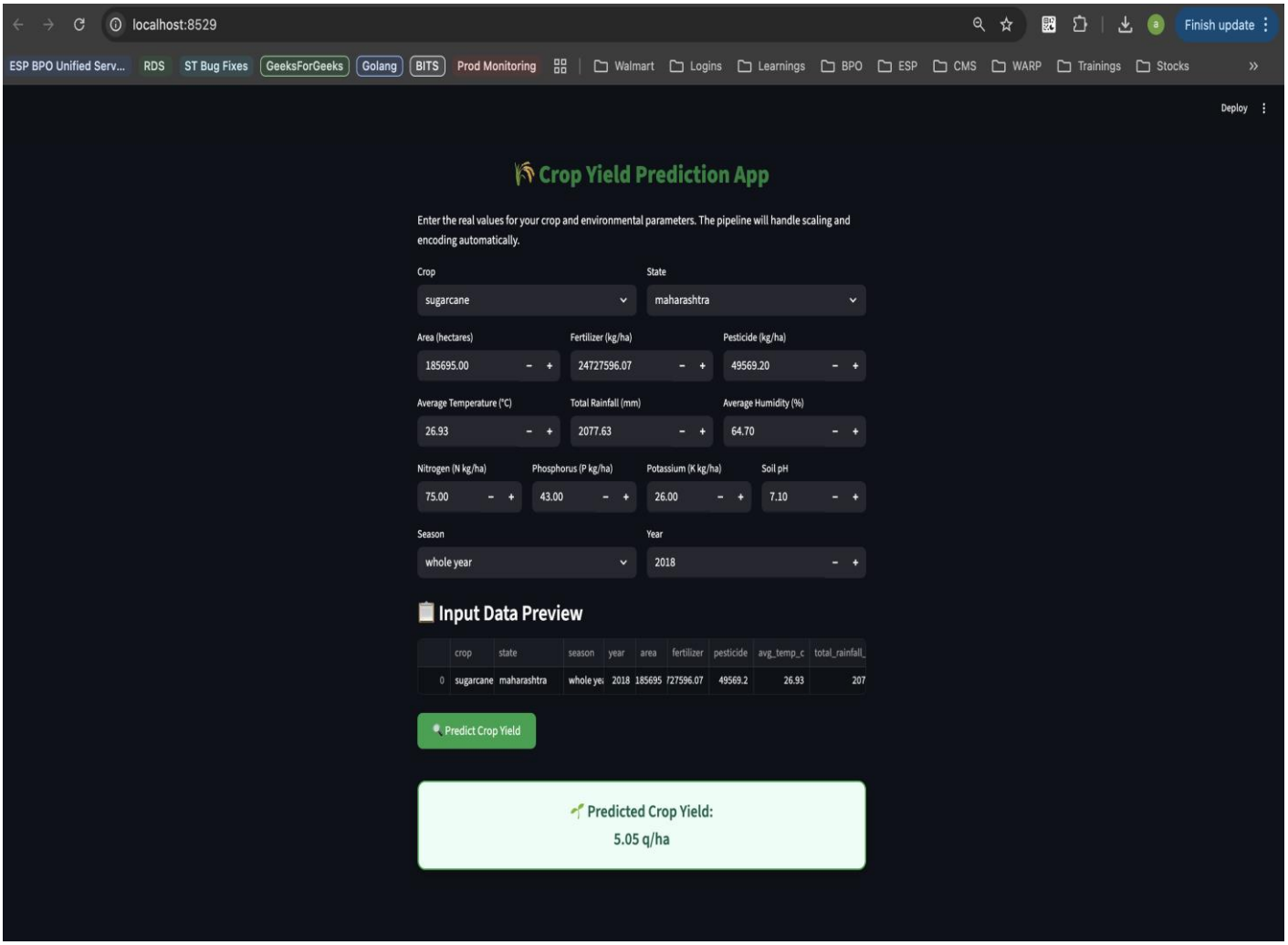
jupyter XGBoost_App.py Last Checkpoint: 3 days ago

File Edit View Settings Help

```python

col1, col2, col3 = st.columns(3)
with col1:
    avg_temp_c = st.number_input("Average Temperature (°C)")
with col2:
    total_rainfall_mm = st.number_input("Total Rainfall (mm)")
with col3:
    avg_humidity_percent = st.number_input("Average Humidity (%)")

col1, col2, col3, col4 = st.columns(4)
with col1:
    n = st.number_input("Nitrogen (N kg/ha)")
with col2:
    p = st.number_input("Phosphorus (P kg/ha)")
with col3:
    k = st.number_input("Potassium (K kg/ha)")
with col4:
    ph = st.number_input("Soil pH")

col1, col2 = st.columns(2)
with col1:
    season = st.selectbox("Season", season_options)
with col2:
    year = st.number_input("Year", min_value=1997, max_value=2025, value=2020, step=1)

# ---------------- Create Input DataFrame ----------------
input_data = pd.DataFrame([{
    'crop': crop_name,
    'state': state_name,
    'season': season,
    'year': year,
    'area': area,
    'fertilizer': fertilizer,
    'pesticide': pesticide,
    'avg_temp_c': avg_temp_c,
    'total_rainfall_mm': total_rainfall_mm,
    'avg_humidity_percent': avg_humidity_percent,
    'n': n,
    'p': p,
    'k': k,
    'ph': ph
}])

# ---------------- Input Data Preview ----------------
st.subheader("📋 Input Data Preview")
st.dataframe(input_data, use_container_width=True)

# ---------------- Prediction ----------------
if st.button("🔍 Predict Crop Yield"):
    try:
        prediction = model_pipeline.predict(input_data)
        st.markdown(f"""
            <div class="result-box">
                🌱 Predicted Crop Yield: <br> <strong>{prediction[0]:.2f} q/ha</strong>
            </div>
        """, unsafe_allow_html=True)
    except Exception as e:
        st.error(f"❌ Error during prediction: {e}")
```

7. Local Deployment of XGBoost_App.py Application Using Streamlit





Project Files Structure

Project Github Link :
https://github.com/swaroopkumaraduru/Swaroop-Kumar-Aduru-Forecasting-Crop-Yield-in-India

1. Data Files
   - crop_yield.csv: Crop-wise yield, area, production, fertilizer, pesticide, rainfall, season, and state (1997–2020).
   - state_soil_data.csv: State-wise soil nutrients (N, P, K) and pH.
   - state_weather_data_1997_2020.csv: State-wise annual weather (temperature, rainfall, humidity).
   - final_merged_statewise_crop_yeild_dataset.csv: Merged dataset combining crop, soil, and weather data for modelling.
   - final_merged_statewise_crop_yeild_dataset.xlsx: Excel version of the merged dataset.
   - cleaned_crop_data.xlsx: Cleaned and preprocessed dataset after outlier handling and normalization.
2. Notebooks
   - Forecasting State-Wise Crop Yield in India_Final.ipynb: Main analysis notebook. Covers:
     - Data loading, merging, and cleaning
     - Exploratory Data Analysis (EDA)
     - Feature engineering and normalization
     - Model building (Linear Regression, Random Forest, XGBoost, etc.)
     - Model evaluation and feature importance
     - Dashboard generation (HTML)
3. Python Apps & Models
   - XGBoost_App.py: Streamlit app for interactive crop yield prediction using the trained XGBoost model.
   - best_model_XGBoost.pkl: Serialized pipeline for XGBoost regression (used by the app).
4. Dashboards & Visualizations
   - state_season_crop_dashboard.html: Interactive dashboard (state-wise seasonal yield and top crops).
   - bivariate_analysis_dashboard.html: Dashboard for bivariate analysis (yield vs. rainfall, temperature, humidity, soil nutrients, crop diversity).
   - Flow_Chart_Data_Merge.png: Visual flowchart showing how datasets are merged.

Key Insights of Project

Data Overview & Patterns

- Crop yield varies significantly across states and seasons.
- Certain crops dominate the dataset, showing higher frequency and production.

Feature Importance

- Features like area, fertilizer, avg_temp_c, and total_rainfall_mm strongly influence yield.
- Seasonal and state-level variations also play a significant role in prediction.

Outlier & Data Quality Analysis

- Outliers exist in yield, area, and production, affecting model accuracy.
- Proper data cleaning and scaling improved model stability.

Model Performance

- Tree-based models like Random Forest and XGBoost performed better than linear models.
- Hyperparameter tuning and cross-validation enhanced predictive performance.

EDA & Visualization Insights

- Boxplots and distribution plots revealed seasonal trends and crop-wise yield differences.
- Correlation analysis highlighted strong relationships between fertilizer usage and yield.

Shortcomings in Data & Model

- Sparse data for some crops or regions limits prediction accuracy.
- Lack of real-time or future weather features restricts forward-looking predictions.

Actionable Predictions

- Model can help farmers estimate expected yield per crop and region.
- Insights can guide optimal fertilizer application and crop planning.

Potential for Dashboard & Decision Support

- Predictions can be visualized in dashboards for state-wise, season-wise, and crop-specific insights.
- Can serve as a decision support tool for policymakers and farmers.

Future Work & Extension or Scope of improvements

Crop-Specific Yield Predictions

- Provide separate yield predictions for each major crop to help farmers plan more effectively.
- Highlight high-risk crops in certain regions based on past performance.

Seasonal Forecasting

- Offer short-term predictions for upcoming seasons to guide sowing and harvesting schedules.
- Include alerts for potential low-yield seasons based on historical trends.

Basic Decision Support Dashboards

- Develop lightweight dashboards using Streamlit or Power BI to visualize:
  - Yield trends over years
  - Top-performing crops and regions
  - Key factors affecting yield (fertilizer, rainfall, temperature)

Predictive Insights for Resource Planning

- Estimate fertilizer or pesticide needs based on predicted yield and crop type.
- Suggest optimal cropping patterns for maximizing productivity on limited land.

Simple Risk Assessment

- Highlight areas prone to low yield due to adverse weather conditions.
- Provide basic "what-if" analysis using existing data (e.g., impact of rainfall shortage on yield).

Data-Driven Recommendations

- Recommend suitable crops for specific states or regions based on historical trends.
- Provide guidelines for crop rotation and seasonal planning.

Interactive Visualization Features

- Add filters for year, state, crop, or season to make dashboards more user-friendly.
- Include charts showing correlations between inputs (fertilizer, rainfall) and yield.

Incremental Model Updates

1. Update predictions regularly as new yearly data becomes available.
2. Keep dashboards relevant without requiring heavy computational resources.

Bibliography

1. India Crop Production. https://www.kaggle.com/datasets/thedevastator/statewise-crop-production-in-india-a-statistical
2. Crop Production Trends https://www.kaggle.com/code/abhijitdahatonde/crop-production-trends

References

1. Overview of Regression Models and How to Determine the Best Model for Data : https://journaljsrr.com/index.php/JSRR/article/view/2452/5060
2. Technology and Management for Social Innovation (IATMSI), Gwalior, India,21–23 December 2022Ranjan, P.; Garg, R.; Rai, J.K. Artificial Intelligence Applications in Soil & Crop Management. In Proceedings of the IEEE Conference on Interdisciplinary Approaches in
3. Gehlot, A.; Sidana, N.; Jawale, D.; Jain, N.; Singh, B.P.; Singh, B. Technical analysis of crop production prediction using MachineLearning and Deep Learning Algorithms. In Proceedings of the International Conference on Innovative Computing, Intelligent.Communication and Smart Electrical Systems (ICSES), Chennai, India, 24–25 September 2022;
3. Vivek, S.; Ashish, K.T.; Himanshu, M. Technological revolutions in smart farming: Current trends, challenges & future directions. Compute. Electron. Agric. **2022**.
4. Mamatha, J.C.K. Machine learning based crop growth management in greenhouse environment using hydroponics farming techniques. Meas. Sens. 2023, 25, 100665.