# Smart Agriculture and Crop Yield Prediction

## Smart Agriculture and Crop Yield Prediction

![63-min.jpg](attachment:e67a3319-0eec-4b9a-a3f5-1fd387d89b03.jpg)

# 1. Problem Statement  
  
Farmers often struggle to predict how much crop they can grow because of changing weather, soil quality, and other factors. This project uses data science to create a simple model that predicts crop yield based on weather, soil, and farming practices.  
  
The goal is to help farmers make better decisions, grow more food, and use their resources like water and fertilizers efficiently.

# 2. Objectives  
  
\*\*1.Boost Crop Production\*\*: Help farmers grow more crops with smart predictions.  
  
\*\*2.Save Time and Money\*\*: Provide tips to reduce waste of water, fertilizers, and effort.  
  
\*\*3.Make Smarter Choices\*\*: Recommend the best crops to grow for higher profits.  
  
\*\*4.Prepare for Challenges\*\*: Warn about risks like droughts or pests in advance.  
  
\*\*5.Support Sustainable Farming\*\*: Promote eco-friendly and efficient farming practices.

# 3. Dataset Information  
![image.png](attachment:7425a227-e66c-4047-b717-3a133d63a3f1.png)  
  
 Where to Get the Dataset: You can download the dataset from   
 claude ai https://www.claude.ai  
  
 And also  
 Download Kaggle https://www.kaggle.com/datasets/datasetengineer/smart-farming-data-2024-sf24  
   
 \*\*1.N, P, K\*\*: Nutrient levels in the soil (Nitrogen, Phosphorus, Potassium).  
  
\*\*2.temperature\*\*: Temperature in the environment (°C).  
  
\*\*\*3.humidity\*\*: Percentage of moisture in the air (%).  
  
\*\*4.ph\*\*: Soil pH level.  
  
\*\*5.rainfall\*\*: Amount of rainfall (mm).  
  
\*\*6.label\*\*: Crop type (target variable).  
  
\*\*7.soil\_moisture\*\*: Soil moisture content (%).  
  
\*\*8.soil\_type\*\*: Categorical representation of soil type.  
  
\*\*9.sunlight\_exposure\*\*: Exposure to sunlight (scale).  
  
\*\*10.wind\_speed\*\*: Wind speed (km/h).  
  
\*\*11.co2\_concentration\*\*: CO₂ levels in the atmosphere (ppm).  
  
\*\*12.organic\_matter\*\*: Organic matter content in the soil.  
  
\*\*13.irrigation\_frequency\*\*: Frequency of irrigation (scale).  
  
\*\*14.crop\_density\*\*: Density of crops in the field.  
  
\*\*15.pest\_pressure\*\*: Pest activity levels.  
  
\*\*17.growth\_stage\*\*: Stage of crop growth (scale).  
  
\*\*18.urban\_area\_proximity\*\*: Distance from urban areas (scale).  
  
\*\*19.water\_source\_type\*\*: Type of water source for irrigation.  
  
\*\*20.frost\_risk\*\*: Risk of frost (scale).  
  
\*\*21.water\_usage\_efficiency\*\*: Efficiency in water usage.

# 4. Code Implementation

## Step 1: Importing Libraries

## Code Snippet

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.model\_selection import cross\_validate  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error  
import warnings  
warnings.filterwarnings('ignore')  
%matplotlib inline

### 2. Load and Explore the Data

## Code Snippet

# Load the dataset  
df = pd.read\_csv('C:/Users/Irfan/Downloads/crop\_data.csv')

## Code Snippet

# Display first few rows  
print("First few rows of the dataset:")  
display(df.head())

## Code Snippet

# Dataset information  
print("\nDataset Info:")  
display(df.info())

## Code Snippet

# Statistical summary  
print("\nStatistical Summary:")  
display(df.describe())

### 3. Data Visualization and Analysis

## Code Snippet

# Create a figure with multiple subplots  
plt.figure(figsize=(15, 10))  
  
# Plot 1: Distribution of crop yields  
plt.subplot(2, 2, 1)  
sns.histplot(df['yield'], bins=20)  
plt.title('Distribution of Crop Yields')  
  
# Plot 2: Yield vs Temperature  
plt.subplot(2, 2, 2)  
sns.scatterplot(data=df, x='temperature', y='yield')  
plt.title('Temperature vs Yield')  
  
# Plot 3: Yield vs Rainfall  
plt.subplot(2, 2, 3)  
sns.scatterplot(data=df, x='rainfall', y='yield')  
plt.title('Rainfall vs Yield')  
  
# Plot 4: Average yield by crop type  
plt.subplot(2, 2, 4)  
sns.barplot(data=df, x='crop\_type', y='yield')  
plt.title('Average Yield by Crop Type')  
plt.xticks(rotation=45)  
  
plt.tight\_layout()  
plt.show()

## Code Snippet

# Correlation heatmap  
plt.figure(figsize=(10, 8))  
numeric\_cols = df.select\_dtypes(include=[np.number]).columns  
sns.heatmap(df[numeric\_cols].corr(), annot=True, cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()

### 4. Data Preprocessing

## Code Snippet

# 1. First, let's add data validation and cleaning  
def clean\_data(df):  
 # Remove any missing values  
 df = df.dropna()  
   
 # Remove outliers using IQR method  
 def remove\_outliers(df, column):  
 Q1 = df[column].quantile(0.25)  
 Q3 = df[column].quantile(0.75)  
 IQR = Q3 - Q1  
 lower\_bound = Q1 - 1.5 \* IQR  
 upper\_bound = Q3 + 1.5 \* IQR  
 return df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]  
   
 # Apply outlier removal to numerical columns  
 numerical\_cols = ['temperature', 'rainfall', 'humidity', 'soil\_ph', 'fertilizer', 'yield']  
 for col in numerical\_cols:  
 df = remove\_outliers(df, col)  
   
 return df

## Code Snippet

# 2. Improved preprocessing function  
def preprocess\_data(data):  
 df\_processed = data.copy()  
   
 # Separate features  
 categorical\_cols = ['soil\_type', 'crop\_type']  
 numerical\_cols = ['temperature', 'rainfall', 'humidity', 'soil\_ph', 'fertilizer']  
   
 # Initialize encoders and scaler  
 label\_encoders = {}  
 scaler = StandardScaler()  
   
 # Handle categorical variables  
 for col in categorical\_cols:  
 label\_encoders[col] = LabelEncoder()  
 df\_processed[col] = label\_encoders[col].fit\_transform(df\_processed[col])  
   
 # Scale numerical features  
 df\_processed[numerical\_cols] = scaler.fit\_transform(df\_processed[numerical\_cols])  
   
 # Also scale the target variable  
 target\_scaler = StandardScaler()  
 df\_processed['yield'] = target\_scaler.fit\_transform(df\_processed[['yield']])  
   
 return df\_processed, label\_encoders, scaler, target\_scaler

## Code Snippet

# 3. Main processing pipeline  
# Load the dataset  
df = pd.read\_csv('crop\_data.csv')  
  
# Clean the data  
df\_cleaned = clean\_data(df)  
  
# Preprocess the data  
df\_processed, label\_encoders, scaler, target\_scaler = preprocess\_data(df\_cleaned)  
  
# Split features and target  
X = df\_processed.drop('yield', axis=1)  
y = df\_processed['yield']  
  
# Split the data with stratification  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y,   
 test\_size=0.2,   
 random\_state=42  
)

## Code Snippet

# 4. Improved model with better hyperparameters  
rf\_model = RandomForestRegressor(  
 n\_estimators=200,  
 max\_depth=10,  
 min\_samples\_split=5,  
 min\_samples\_leaf=2,  
 max\_features='sqrt',  
 random\_state=42,  
 n\_jobs=-1  
)

## Code Snippet

# 5. Model training with cross-validation  
  
cv\_results = cross\_validate(  
 rf\_model,  
 X\_train,  
 y\_train,  
 cv=3,  
 scoring=('r2', 'neg\_mean\_squared\_error', 'neg\_mean\_absolute\_error'),  
 return\_train\_score=True  
)

### 5. Model Training and Evaluation

## Code Snippet

# Train the final model  
rf\_model.fit(X\_train, y\_train)

## Code Snippet

# Make predictions  
y\_pred = rf\_model.predict(X\_test)

## Code Snippet

# Initialize and train the Random Forest model  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
rf\_model.fit(X\_train, y\_train)

## Code Snippet

# Calculate metrics  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print("Model Performance Metrics:")  
print(f"Root Mean Squared Error: {rmse:.2f}")  
print(f"Mean Absolute Error: {mae:.2f}")  
print(f"R² Score: {r2:.2f}")

## Code Snippet

cv\_scores = cross\_val\_score(rf\_model, X, y, cv=4, scoring='r2')  
print(f"\nCross-validation R² scores: {cv\_scores}")  
print(f"Average CV R² score: {cv\_scores.mean():.2f}")

### 6. Feature Importance Analysis

## Code Snippet

# Calculate and plot feature importance  
feature\_importance = pd.DataFrame({  
 'feature': X.columns,  
 'importance': rf\_model.feature\_importances\_  
}).sort\_values('importance', ascending=False)  
  
plt.figure(figsize=(10, 6))  
sns.barplot(data=feature\_importance, x='importance', y='feature')  
plt.title('Feature Importance in Crop Yield Prediction')  
plt.show()

### 7. Prediction Function for New Data

## Code Snippet

def predict\_yield(new\_data, label\_encoders, scaler):  
 """  
 Make predictions for new data  
 """  
 # Create a copy of the input data  
 data = new\_data.copy()  
   
 # Encode categorical variables  
 categorical\_cols = ['soil\_type', 'crop\_type']  
 numerical\_cols = ['temperature', 'rainfall', 'humidity', 'soil\_ph', 'fertilizer']  
   
 for col in categorical\_cols:  
 data[col] = label\_encoders[col].transform(data[col])  
   
 # Scale numerical features  
 data[numerical\_cols] = scaler.transform(data[numerical\_cols])  
   
 # Make prediction  
 prediction = rf\_model.predict(data)  
   
 return prediction  
  
# Example usage  
new\_data = pd.DataFrame({  
 'temperature': [26],  
 'rainfall': [160],  
 'humidity': [68],  
 'soil\_ph': [6.8],  
 'soil\_type': ['loamy'],  
 'fertilizer': [105],  
 'crop\_type': ['wheat']  
})  
  
prediction = predict\_yield(new\_data, label\_encoders, scaler)  
print(f"Predicted yield: {prediction[0]:.2f}")

### 8. Model Performance Visualization

## Code Snippet

# Create visualization of actual vs predicted values  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_test, y\_pred, alpha=0.5)  
plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2)  
plt.xlabel('Actual Yield')  
plt.ylabel('Predicted Yield')  
plt.title('Actual vs Predicted Crop Yield')  
plt.tight\_layout()  
plt.show()

## Code Snippet

# Plot residuals  
residuals = y\_test - y\_pred  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_pred, residuals, alpha=0.5)  
plt.xlabel('Predicted Yield')  
plt.ylabel('Residuals')  
plt.axhline(y=0, color='r', linestyle='--')  
plt.title('Residual Plot')  
plt.tight\_layout()  
plt.show()

## Step 6: Conclusion  
Smart Agriculture and Crop Yield Prediction use data and technology to help farmers make better decisions. By understanding soil, weather, and crop needs, this system helps farmers:  
  
Choose the right crops.  
  
Increase production.  
  
Save resources like water and fertilizers.  
  
This makes farming easier, more profitable, and good for the environment, ensuring enough food for everyone.

![WhatsApp Image 2025-01-10 at 20.00.20\_226b8099.jpg](attachment:0a39f2e6-4220-4bc4-9ad0-6adb822ee5dd.jpg)

## Code Snippet