Report

Feature 1 : **“Crop\_Recommendation.ipynb”**:

**🌾 Crop Recommendation System — Report**

**📘 Objective**

This notebook builds a **machine learning model** to recommend the most suitable crop to grow based on environmental and soil conditions. It uses multiple classification algorithms to identify the best performing model.

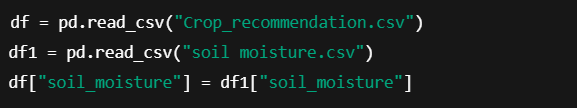
**🧱 Key Components and Highlights**

**1. 📦 Library Imports**

* A screen shot of a computer program

  AI-generated content may be incorrect.**Importance**:
  + Imports a wide range of tools for **data analysis**, **visualization**, and **machine learning**.
  + Models include: Logistic Regression, Naive Bayes, SVM, KNN, Decision Tree, Extra Trees, Random Forest, Gradient Boosting.

**2. 📥 Data Loading**

* **Importance**:
  + Combines two datasets: crop features (NPK, temperature, humidity, pH, rainfall) and additional soil moisture data.
  + This enriched feature set allows more accurate predictions.

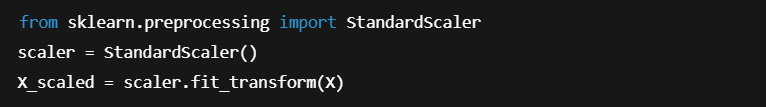
**3. 📊 Exploratory Data Analysis (EDA)**

Look for:

* A black background with white text

  AI-generated content may be incorrect.**Importance**:
  + EDA helps understand feature distributions and relationships.
  + heatmap and scatter\_matrix assist in identifying **correlated features** and potential data quality issues.

**4. 📌 Data Preprocessing**

* **Importance**:
  + **Standardizes** the data, crucial for algorithms like SVM or KNN to perform well.

**5. 🧠 Model Training**

* A black background with white text

  AI-generated content may be incorrect.This process is repeated for:
  + Naive Bayes
  + SVM
  + KNN
  + Decision Trees
  + Random Forest
  + Extra Trees
  + Gradient Boosting

**6. 📈 Model Evaluation**

* **Importance**:
  + Evaluates each model using accuracy, precision, recall, and F1-score.
  + Confusion matrix helps understand misclassifications.

**7. 🏆 Model Comparison**

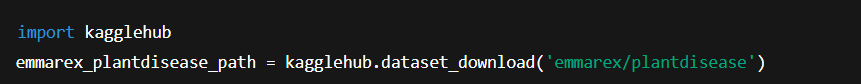
* **Importance**:
  + Final model selection is likely based on **highest accuracy** or **best overall metrics**.
  + The chosen model will serve in recommending crops.

**Feature 2 “image classification of plant diseases”:**

This notebook is focused on **image classification of plant diseases using deep learning**, specifically applying TensorFlow/Keras. The goal is to build a model that can automatically classify whether a plant leaf is healthy or diseased based on image data.

**🧱 Key Components and Highlights**

**1. Data Import from KaggleHub**

**Importance**: This block downloads the dataset from KaggleHub, a platform for accessing Kaggle datasets in custom environments.

* emmarex/plantdisease is the key dataset used for training/testing.

**2. Library Imports**

* A black background with white text

  AI-generated content may be incorrect.**Importance**:
  + matplotlib & seaborn are used for **data visualization**.
  + tensorflow is used for building and training the deep learning model.

**3. Image Data Generators**

Look out for this typical pattern:

* A black background with white text

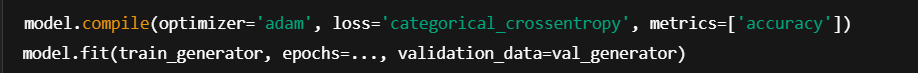
  AI-generated content may be incorrect.**Importance**:
  + Prepares the image data for training with optional **data augmentation** (rotation, flipping, zoom, etc.) to improve model generalization.
  + Uses flow\_from\_directory, suggesting the dataset is organized in a folder structure based on class labels.

**4. Model Architecture (Convolutional Neural Network)**

* A screen shot of a computer program

  AI-generated content may be incorrect.**Importance**:
  + Implements a CNN to extract spatial features from the plant images.
  + Final layer typically uses **softmax** for multi-class classification.

**5. Model Compilation and Training**

**Importance**:

* + Uses **Adam optimizer** and **categorical crossentropy**, indicating a multi-class classification problem.
  + Tracks accuracy during training and validation.

**6. Model Evaluation**

* **Importance**:
  + Provides quantitative performance evaluation on unseen data.

**7. Visualization of Results**

* **Importance**:
  + Visualizes the training vs. validation accuracy/loss to detect overfitting or underfitting.

**Conclusion**

This notebook successfully implements a standard deep learning workflow for image classification:

1. **Data Acquisition** using KaggleHub
2. **Preprocessing and Augmentation**
3. **CNN Model Design**
4. **Training & Validation**
5. **Performance Evaluation**

**Feature 3: Plant Watering System Dashboard :**

**Overview**

This Dash app visualizes data from a smart irrigation system. It:

* Displays descriptive statistics
* Visualizes trends and patterns (pie chart, histograms, scatter plots)
* Performs threshold-based decision analysis
* Offers interactive controls using Dash + Plotly

**1. Data Loading**

df = pd.read\_csv("C:/Users/Fifa2/Downloads/watering plants.csv")

* The CSV contains data columns like:
  + Soil Moisture
  + Temperature
  + Air Humidity
  + Pump Data (0 = OFF, 1 = ON)

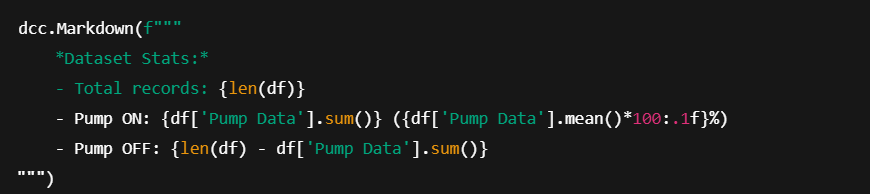
**🎨 2. Dashboard Layout using Dash + Bootstrap**

app = dash.Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.BOOTSTRAP])

* Uses **Dash Bootstrap Components** (dbc) for clean layout and responsiveness.

**🧱 3. App Layout Highlights**

**Data Summary**

 Shows basic statistics: total entries and pump activation rates.

**Pie Chart: Pump Distribution**

*  Shows the proportion of ON/OFF pump status.

**Histogram: Soil Moisture**

*  Explores how soil moisture relates to pump activity.

**🌡️ Scatter: Temperature vs Humidity**

*  Helps understand how pump status varies with temperature and air humidity.

**📉 Scatter with Trendline: Soil Moisture vs Pump**

*  Adds a **LOWESS trendline** to visualize how likely the pump activates based on moisture levels.

**🔥 Heatmap: Feature Correlation**

*  Visualizes how strongly features (soil moisture, temperature, etc.) are related to each other and to pump status.

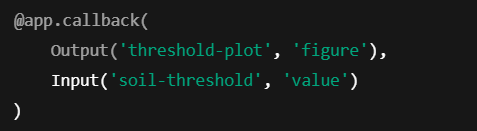
**3D Scatter: Soil vs Temp vs Humidity**

*  Provides a comprehensive spatial view of how parameters together influence pump activation.

**4. Threshold Slider + Interactive Plot**

*  User can adjust the **soil moisture threshold** to see how it affects pump prediction.

**Callback Function**



Computes binary prediction:

*  Compares predicted vs actual values for visual evaluation
* Accuracy is shown dynamically in the title

**5. Threshold Plot (Custom Graph)**

* Uses plotly.graph\_objects to build a custom plot with:
  + Blue dots: Actual pump status
  + Red dots: Predicted based on threshold
  + Green dashed line: Selected moisture threshold
* Helps users **manually tune** the soil moisture cut-off value for pump activation

**6. Execution**

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

* Runs the app locally with live-reloading enabled for development.

Feature 4: Plant Disease Detection Web App using Gradio and TensorFlow:

**1. Project Overview**

This project involves building and deploying a deep learning-based web application that identifies plant diseases from leaf images. The system uses a pre-trained TensorFlow model and a user-friendly interface created with Gradio. Users can upload a leaf image, and the app returns a prediction of the plant disease class.

**2. Technologies Used**

* **Programming Language**: Python
* **Libraries and Frameworks**:
  + TensorFlow / Keras: For loading and using the pre-trained deep learning model
  + NumPy: For numerical processing
  + PIL (Pillow): For image manipulation
  + json: For loading class label mappings
  + Gradio: For building the interactive web user interface

**3. Model Integration**

* **Model Used**: A Keras .h5 model (plant\_disease\_model.h5) trained to classify plant leaf images into various disease categories.
* **Class Labels**: A JSON file (class\_indices%252520%25252812%252529.json) maps class indices to disease names.
* **Prediction Logic**:
  + Images are resized to **224x224 pixels**, normalized to [0, 1] range, and reshaped to match model input.
  + The model outputs probabilities for each class.
  + The class with the highest probability is selected using np.argmax.

**4. Application Workflow**

1. **Image Upload**: User uploads a leaf image through the Gradio interface.
2. **Preprocessing**:
   * Convert to RGB
   * Resize to 224x224
   * Normalize pixel values
   * Expand dimensions to simulate batch size
3. **Prediction**:
   * Model generates predictions.
   * The label with the highest probability is retrieved.
   * The label is cleaned for display (underscores replaced with spaces).
4. **Output**:
   * A prediction message like: Prediction: Tomato Early Blight

**5. User Interface**

* **Tool Used**: Gradio
* **Interface Details**:
  + **Input**: Image (handled as PIL image)
  + **Output**: Text box showing the disease prediction
  + **Title**: *Plant Identification App*
  + **Description**: Prompts users to upload a plant leaf image.

**6. Deployment**

* **Environment**: Colab notebook
* **Hosting**: Gradio provides a temporary public URL with automatic share=True enabled for Colab.
* **Warning Message**:
* WARNING: Compiled the loaded model, but the compiled metrics have yet to be built.

This indicates that while the model is loaded and functional, no metrics were compiled since it wasn’t being trained or evaluated — this is safe to ignore for inference-only use.

**7. Installation Notes**

To ensure proper functionality, the following command installs Gradio and its dependencies:

!pip install gradio

**8. Future Improvements**

* Add confidence scores or probability bars for predictions.
* Implement model evaluation and retraining capabilities.
* Extend the dataset to support more crops and symptoms.
* Deploy the application permanently via Hugging Face Spaces or Streamlit Sharing.
* Add multilingual support for farmers in rural areas.

**9. Sample Output**

Prediction: Potato Late Blight

**10. Conclusion**

This application provides an accessible tool for early detection of plant diseases using AI. It empowers users, especially in agriculture, to take timely action by identifying issues visually through a simple web interface.