**Predicting Plant Health Using Machine Learning Models**

**Week 4 Update**

**Introduction:**

* In this document, I aim to explain the predictive model that monitors and analyzes plant health based on various environmental parameters. This initiative is a part of the smart planter project which is designed to assist both home gardeners and farmers in optimizing their yield and minimizing waste. The predictive model will utilize machine learning algorithms to analyze data from various sensors and predict the health of the plant.

**Dataset Description:**

* The dataset contains the following attributes:
  + **Humidity (%):** The percentage of water vapor present in the air surrounding the plants.
  + **Nitrogen Levels (mg/kg):** The concentration of nitrogen in the soil, measured in milligrams per kilogram.
  + **Ambient Temperature (°C):** The surrounding temperature measured in degrees Celsius.
  + **Presence of Sunlight**: A binary attribute indicating whether the plant is exposed to sunlight.
  + **Plant Health:** The target variable indicating the health status of the plant.

**Objective**

* To develop a predictive model that accurately predicts plant health based on the given features.
* To assist in the development of a smart planter system that is user-friendly and provides real-time insights into plant health.

**Tools Uses For this project:**

1. **Google Colab:**
   * **Free Cloud Environment:** Google Colab provides free access to a cloud based Jupyter notebook environment with GPU support. This is extremely useful for individuals and small teams that may not have access to high-performance hardware for training large ML models.
   * **Pre-installed Libraries:** Colab comes with many pre-installed libraries commonly used in ML, such as TensorFlow, PyTorch, and scikit-learn. This saves time on environment setup.
   * **Easy Sharing and Collaboration:** You can easily share Colab notebooks with others, making it a great tool for collaborative work. Multiple people can work on the same notebook simultaneously.
   * **Integration with Google Drive:** Colab integrates seamlessly with Google Drive, making it simple to save and organize your notebooks and datasets.
   * **Access to Datasets:** Colab offers access to various datasets through Google Cloud, which can be helpful for experimenting with different data sources.
   * **Uses:** It is simple to use, just go to the website <https://colab.research.google.com/> and then click on the upload section and upload the ipynb file from the local computer or can create a new ipynb file.
2. **Visual Studio Code (VS Code) with Jupyter Notebook:**
   * **Customizable Development Environment:** VS Code is a highly customizable code editor. With the Jupyter extension, you can turn it into a full-fledged Jupyter Notebook environment. This allows you to use your preferred code editor with all the features and extensions you need.
   * **Local Development:** Unlike Colab, which is cloud-based, VS Code is a local development environment. This means you have more control over your environment and can work offline.
   * **Integrated Debugging:** VS Code offers powerful debugging capabilities, which can be extremely useful when developing complex ML models. You can set breakpoints, inspect variables, and debug step by step.
   * **Rich Ecosystem:** VS Code has a rich ecosystem of extensions, including those for Git integration, code linting, and many other development tasks. This makes it versatile for both ML and general software development.
   * **Code Version Control:** With VS Code, you can easily integrate with version control systems like Git, which is crucial for tracking changes in your ML projects.
   * **Uses:** To follow the local environment, there is need to setup the VS code and Ipynb in VS code and then open the ipynb file and run it.

The choice between Google Colab and VS Code with Jupyter Notebook integration often depends on the specific needs and preferences:

* Use Google Colab if you want a hassle-free, cloud-based environment with GPU support, easy collaboration, and access to various datasets.
* Use VS Code with Jupyter Notebook if you prefer a customizable local environment, require advanced debugging capabilities, or work on ML projects that involve a mix of code development and data analysis. For further installation and its configuration, you may follow this article link: <https://www.alphr.com/vs-code-open-jupyter-notebook/>
* Currently I am using VS Code with Jupyter Notebook, but if I run the same code with google colab there will be no issue with it.

**Data Simulation Function: generate\_data**

* This function simulates the generation of a dataset with 10000 data points, which are equally distributed across three categories: 'Healthy', 'Moderate', and 'Unhealthy'. These categories represent different states of plant health. The attributes for each data point include 'Humidity (%)', 'Nitrogen Levels (mg/kg)', 'Ambient Temperature (°C)', 'Presence of Sunlight', and 'Plant Health'. The values of these attributes are generated based on certain conditions defined for each category to simulate real-world scenarios.
* **Parameters**
  + **n**: The total number of data points to be generated. It is set to 10000 by default.
* **Returns**
  + **df**: A shuffled pandas DataFrame containing the generated data points with their respective attributes.

**Function Breakdown**

1. **Initialization**:
   * A dictionary named **data** is initialized with keys representing the different attributes and empty lists as their corresponding values.
2. **Data Generation**:
   * The function iterates over three categories: 'Healthy', 'Moderate', and 'Unhealthy'.
   * For each category, it generates **n/3** data points (to ensure an equal distribution) with attribute values generated based on predefined conditions that represent typical values for each category.
3. **Conditions for Data Generation**:
   * **Healthy**:
     + Humidity: Between 50% and 70%.
     + Nitrogen Levels: Between 100 and 150 mg/kg.
     + Ambient Temperature: Between 15°C and 25°C.
     + Presence of Sunlight: Always 1 (present).
     + Plant Health: Labeled as 'Healthy'.
   * **Moderate**:
     + Humidity: Between 40% and 80%.
     + Nitrogen Levels: Between 80 and 170 mg/kg.
     + Ambient Temperature: Between 10°C and 30°C.
     + Presence of Sunlight: Randomly chosen between 0 (absent) and 1 (present).
     + Plant Health: Labeled as 'Moderate'.
   * **Unhealthy**:
     + Humidity: Between 0% and 100%.
     + Nitrogen Levels: Between 0 and 200 mg/kg.
     + Ambient Temperature: Between -10°C and 40°C.
     + Presence of Sunlight: Randomly chosen between 0 (absent) and 1 (present).
     + Plant Health: Labeled as 'Unhealthy'.
4. **Dataframe Creation and Shuffling**:
   * After generating the data, it is converted into a panda DataFrame.
   * The DataFrame is then shuffled to randomize the order of data points and the index is reset.

* To create a balanced dataset with an equal number of 'Healthy', 'Moderate', and 'Unhealthy' instances, I create a function **generate\_data** to simulate the dataset with each label, and then shuffle the dataset to mix it.
* **Explanation**:
  + I divide the total number of data points by 3 to get an equal number of data points for each category ('Healthy', 'Moderate', and 'Unhealthy').
  + I then generate data for each category separately, using different ranges of values for the features to simulate the different conditions for each category.
  + After generating the data, I create a DataFrame and shuffle it using df.sample(frac=1).reset\_index(drop=True) to mix the data points from the different categories together.
* Here is the result, showing top 10 rows of dataset:

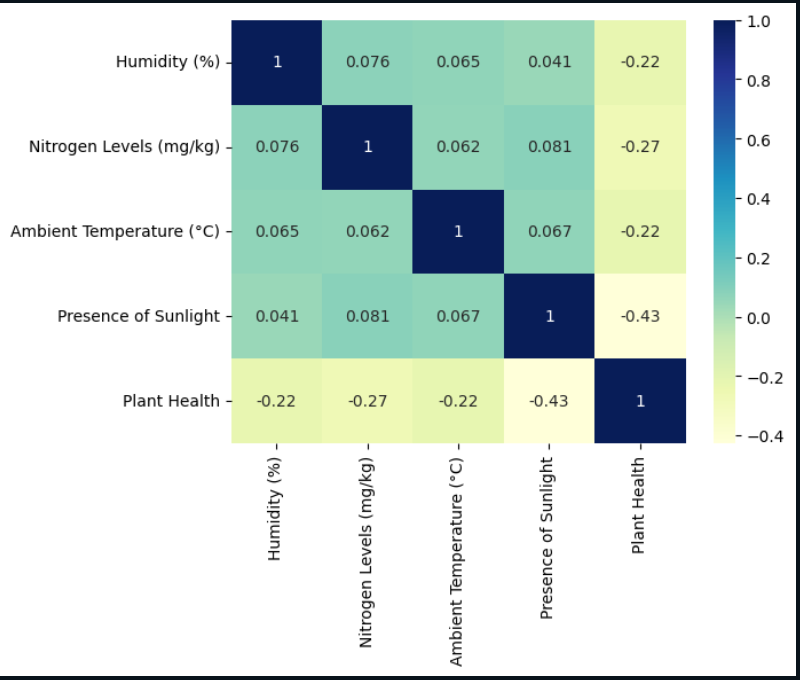
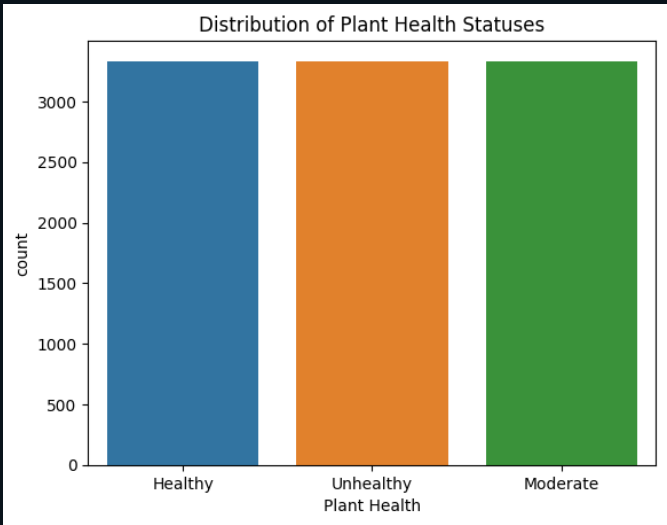
A screenshot of a computer screen

Description automatically generated

**Data Preprocessing:**

* Perform necessary data preprocessing steps.
* For instance, handling missing values, encoding categorical variables, etc.
* Like in the given dataset we have plant health status in the strings like Healthy, Unhealthy and Moderate, I encoded it and assigned numeric values like 0, 2 and 1 respectively, because it is easy for model to train with numeric values, and as other data in the numeric form therefore I update the plant health status to numeric values. After model prediction I will decode the values and will reassign the strings to the respective values.

**Exploratory Data Analysis:**

* Perform some exploratory data analysis on the dataset generated.
* For instance, using seaborn or matplotlib to visualize the data.
* Correlation between different features: Below figure shows the visualization of correlations among different features.
* 
* In the above correlation matrix figure, I observe the relationships between various environmental factors and plant health. The "Humidity (%)" has a weak positive correlation with "Nitrogen Levels (mg/kg)", "Ambient Temperature (°C)", and "Presence of Sunlight", indicating that as one of these factors increases, the other tends to slightly increase as well. However, it has a weak negative correlation with "Plant Health", suggesting that higher humidity levels might be associated with a decline in plant health. Similarly, "Nitrogen Levels (mg/kg)" also exhibit a weak positive correlation with "Ambient Temperature (°C)" and "Presence of Sunlight", and a more noticeable negative correlation with "Plant Health", implying that higher nitrogen levels might be linked to poorer plant health. "Ambient Temperature (°C)" shows a very weak positive correlation with "Presence of Sunlight". The "Presence of Sunlight" has a moderate negative correlation with "Plant Health", indicating that increased sunlight exposure might be significantly associated with deteriorating plant health. Overall, it seems that higher levels of these environmental factors are somewhat associated with a decline in plant health, with sunlight having the most substantial negative impact. It's important to note that these correlations are generally weak, suggesting that the relationships between these variables are not very strong, and other factors might be influencing plant health as well.
* Count Plot of Plant Health Statuses: Below figure shows the count of existing healthy, unhealthy, and moderate classes of dataset.
* 
* The above Plot represents that we have balanced dataset.

**Model Development:**

In this section, I will develop multiple machine learning models and will compare the results of predict plant health based on the features provided.

**Splitting & Scaling the Data:** I split the dataset into 80 percent training and 20 percent testing.

**Feature Scaling:** Normalize the dataset using standard scalar and fit transform function.

**Model 1: Building Neural Networks**

* Developed a Neural Network model to predict plant health based on various environmental factors. Neural Networks are a category of algorithms modeled loosely after the human brain. They are designed to recognize patterns in complex data, and with multiple layers, they can capture intricate relationships in the dataset.
* I started by setting up a neural network structure with several layers, including input layers, hidden layers, and an output layer. The activation functions, dropout rates, and other parameters will be fine-tuned to optimize the model's performance. I will also be implementing techniques such as batch normalization to stabilize and possibly accelerate the learning process.
* After building the model, I compiled it using an appropriate optimizer and loss function, followed by training the model with our training dataset. I also validate the model using a separate validation dataset to avoid overfitting and to ensure that our model generalizes well to new, unseen data.
* Finally, I evaluated the model's performance using various metrics such as accuracy and loss over the epochs and visualized these metrics to analyze the model's learning curve and to make necessary adjustments for improvement.

**Model 2: K-Nearest Neighbors (KNN) Classifier Model**

* A K-Nearest Neighbors (KNN) Classifier model to predict plant health based on various environmental parameters such as humidity percentage, nitrogen levels, ambient temperature, and the presence of sunlight. The KNN algorithm is a type of instance-based learning that classifies a data point based on the majority class of its 'K' nearest neighbors in the feature space.
* Initially, I prepared the data by segregating the features and the target variable, followed by splitting the data into training and testing sets. Scaling the features is a crucial step since KNN is a distance-based algorithm, and having features on a similar scale helps in improving the model's performance.
* Next, I initialized the KNN classifier, specifying a suitable number of neighbors (K) to consider. I plan to experiment with different values of 'K' to find the optimal number that yields the best performance without overfitting. The model will then be trained using the training dataset.
* Subsequently, I used the trained model to make predictions on the testing data. The model's performance is evaluated using various metrics such as accuracy, precision, and recall, which will be presented in a detailed classification report.

**Model 3: Training a Tuned Logistic Regression Model with Class Weights**

* I developed a Logistic Regression model, fine-tuned with the integration of class weights, to predict the health of plants based on the given environmental attributes: humidity percentage, nitrogen levels, ambient temperature, and the presence of sunlight. Logistic Regression, a statistical method for analyzing datasets where the outcome variable is categorical, is particularly suitable for binary or multiclass classification tasks.
* To begin, I segregated the dataset into features and the target variable, followed by partitioning the data into training and testing subsets. This step ensures that the model can be evaluated on unseen data to gauge its predictive accuracy and generalization capabilities.
* Next, I initialized the Logistic Regression classifier, incorporating class weights into the model. The inclusion of class weights helps in handling any imbalance in the dataset by assigning different weights to each class, thus preventing the model from being biased towards the majority class. I will also fine-tune other hyperparameters to optimize the model's performance further.
* After setting up the model, I proceed to train it using the training dataset. This step involves learning the underlying patterns in the data to make accurate predictions.
* Following the training phase, I used the model to make predictions on the testing data and evaluate its performance using various metrics such as accuracy, precision, and recall. These metrics will be detailed in a comprehensive classification report, providing insights into the model's performance across different classes.

**Model 4: Training a Tuned Random Forest Model with Class Weights**

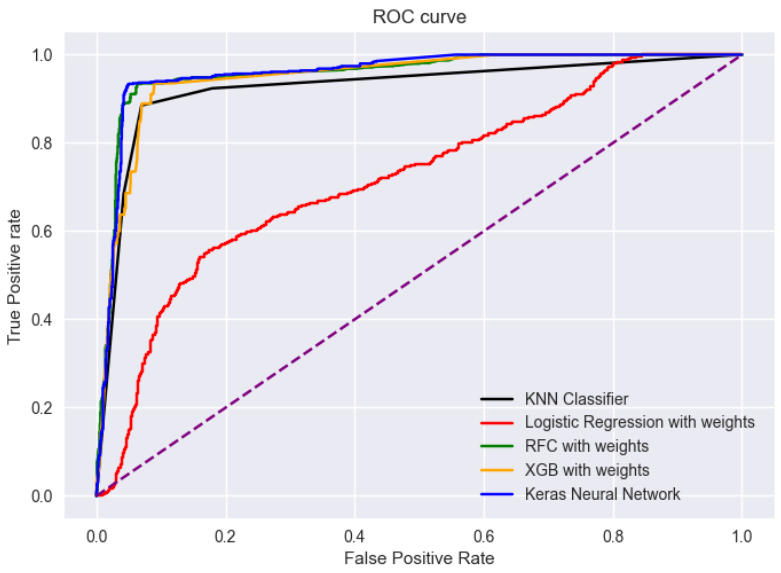
* I focused on developing a Random Forest model, a powerful ensemble learning method, fine-tuned with class weights, to predict plant health based on the analyzed environmental factors: humidity percentage, nitrogen levels, ambient temperature, and the presence of sunlight. The Random Forest algorithm, which operates by constructing multiple decision trees during training time and outputting the class that is the mode of the classes from individual trees, is known for its high accuracy, ability to handle large data sets with higher dimensionality, and its ability to handle missing values.
* First, I segregated the data into features and the target variable, and then partitioned it into training and testing sets to validate the model's performance on unseen data later.
* Next, I initialized the Random Forest classifier, incorporating class weights to address any potential class imbalance in the dataset. This strategy ensures that the model does not exhibit a bias towards the majority class, providing a balanced approach to classification. Additionally, I will fine-tune various hyperparameters such as the number of trees and the maximum depth of the trees to optimize the model's predictive performance.
* Following the initialization, I trained the model using the training dataset, allowing it to learn the complex patterns and relationships in the data.
* After the training phase, I employed the model to make predictions on the testing data, subsequently evaluating its performance using several metrics including accuracy, precision, and recall. A detailed classification report will be generated to provide a comprehensive view of the model's performance across different classes.

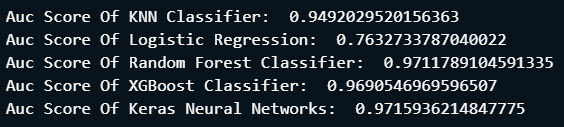
**Model 5: Training a Tuned XGBoost Classifier Model with Class Weights**

* I will be crafting an XGBoost Classifier model, fine-tuned with class weights, to predict plant health based on the designated environmental factors: humidity percentage, nitrogen levels, ambient temperature, and the presence of sunlight. XGBoost, which stands for eXtreme Gradient Boosting, is an implementation of gradient boosted decision trees designed for speed and performance.
* Initially, I delineated the dataset into features and the target variable, followed by a division into training and testing sets. This division is vital to assess the model's efficacy on unseen data, ensuring a reliable evaluation of its predictive capabilities.
* Subsequently, I initialized the XGBoost classifier, integrating class weights to address any imbalances in the class distribution within the dataset. This integration ensures a balanced approach to the classification task, preventing the model from favoring the majority class and potentially overlooking the minority class. Moreover, I will fine-tune various hyperparameters such as learning rate and max depth to enhance the model's performance further.
* Once the model is set up, I trained it using the training dataset, allowing it to learn and adapt to the underlying patterns in the data, which will be instrumental in making accurate predictions.
* After the training process, I utilized the model to make predictions on the testing data. The performance of the model will be evaluated using a range of metrics including accuracy, precision, and recall, offering a detailed insight into its classification prowess across different classes. A comprehensive classification report will be generated to encapsulate these insights.

**Performance Comparison Between Above Models Using ROC AUC Curve:**

* I illustrated the performance of all the developed models by plotting their multi-class Receiver Operating Characteristic (ROC) curves alongside their respective Area Under the Curve (AUC) scores. In a multi-class classification scenario, the ROC curves represent the true positive rate (TPR) versus the false positive rate (FPR) for each class, considering it as the positive class while grouping the rest as the negative class.
* The AUC scores, which are computed for each class separately, provide a quantitative measure of a model's ability to distinguish between the classes across different thresholds. These scores are particularly vital in understanding how well the models can differentiate between the three distinct plant health categories based on the given environmental parameters.
* Each curve in the plot represents a different model, with the colors of the curves being allocated according to the AUC scores. This color-coding scheme facilitates a quick and clear comparison of the models, helping in identifying the ones with superior performance, indicated by higher AUC scores.
* Through this visualization, I aim to discern the model that demonstrates the best performance in terms of sensitivity and specificity for each class in the multi-class classification problem. This analysis will be instrumental in selecting the most optimal model for predicting plant health, ensuring a reliable and accurate classification.
* Bellow graph shows the ROC AUC curve:



****

**Why and Why Not Consider the model:**

In this project, I embarked on the journey to develop a predictive model capable of accurately determining plant health based on various environmental parameters. Through rigorous analysis and model development, I have explored several machine learning and deep learning models, each with its unique strengths and weaknesses. Here, I will discuss each model in the context of their performance as illustrated by the ROC-AUC curves and scores, and provide insights into the reasons for selecting or not selecting them for the final implementation:

1. **KNN Classifier (AUC = 0.949)**
   * **Why Choose**: The KNN Classifier demonstrated a commendable performance with an AUC score close to 0.95, indicating a good ability to distinguish between the different classes of plant health. Its relatively simple algorithm and high AUC score make it a strong contender.
   * **Why Not Choose**: Despite its good performance, it falls slightly short when compared to other models with higher AUC scores. Moreover, KNN can be computationally intensive with larger datasets.
2. **Logistic Regression with Weights (AUC = 0.763)**
   * **Why Choose**: Logistic regression offers a straightforward and interpretable model, which can be beneficial in understanding the relationships between the features and the target variable.
   * **Why Not Choose**: Unfortunately, this model exhibited the lowest AUC score among all the models tested, indicating a lesser ability to correctly classify the plant health categories. This makes it a less favorable choice for this particular task.
3. **Random Forest Classifier with Weights (AUC = 0.971)**
   * **Why Choose**: The Random Forest Classifier showcased an excellent performance with an AUC score nearing 0.97. Its ensemble nature, which combines multiple decision trees to improve predictive accuracy, makes it a robust choice for this task.
   * **Why Not Choose**: Despite its high performance, Random Forest models can sometimes be complex and less interpretable compared to simpler models, which might pose challenges in explaining the model predictions.
4. **XGBoost Classifier with Weights (AUC = 0.969)**
   * **Why Choose**: The XGBoost model, with an AUC score close to 0.97, stands as a powerful tool for this classification task. Its gradient boosting framework is known for delivering high predictive accuracy.
   * **Why Not Choose**: Like the Random Forest model, the complexity of the XGBoost model might make it less interpretable, which could be a drawback if model interpretability is a priority.
5. **Keras Neural Network (AUC = 0.972)**
   * **Why Choose**: The Neural Network model achieved the highest AUC score among all the models, indicating an excellent ability to distinguish between the different plant health categories. Its deep learning framework allows it to capture complex patterns in the data, making it a highly potent choice for this task.
   * **Why Not Choose**: Neural networks are often considered as "black box" models, making them less interpretable. Moreover, they require a substantial amount of data for training and can be computationally intensive.

**Recommended Model for the data coming from sensors:**

Based on the analysis and the AUC scores obtained from the ROC-AUC curves, I would recommend the **Keras Neural Network model** for predicting plant health using data coming from the sensors. Here are the reasons for this recommendation:

1. **Highest AUC Score (0.972)**: The Keras Neural Network model achieved the highest AUC score among all the models tested, indicating its superior ability to distinguish between the different classes of plant health accurately. This high score suggests that the model has learned complex patterns and relationships in the data, which is vital for making accurate predictions.
2. **Deep Learning Advantage**: Neural networks, being a subset of deep learning, can learn and model non-linear and complex relationships, which might be prevalent in sensor data. This makes them particularly adept at handling complex datasets and extracting subtle patterns that might be missed by other models.
3. **Adaptability to Sensor Data**: Neural networks can be fine-tuned further to adapt to the specific characteristics of sensor data. With the right preprocessing and feature engineering, the neural network model can potentially offer even higher predictive accuracy.
4. **Potential for Further Optimization**: Neural networks offer a wide range of hyperparameters and architectures that can be explored and optimized further to enhance performance. With more data and further tuning, the performance of this model can potentially be improved even further.

**Development of Plant Health Prediction API**

The primary objective for this week was to develop an API that can predict plant health based on environmental variables. The API is designed to receive environmental parameters as input and return the predicted health status of the plant.

**Development Process:**

A Flask application was developed to serve as the API, utilizing a previously trained Neural Network model. The model was trained to predict plant health based on four environmental parameters: humidity, nitrogen level, ambient temperature, and the presence of sunlight. The model and the scaler, used for normalizing the input data, were serialized using **pickle** and loaded into the Flask application.

**API Endpoints:**

The API has two main endpoints:

1. **Home Endpoint ('/'):** This is a simple endpoint returning a string message, indicating that the server is active and ready to provide plant health care services.
2. **Predict Endpoint ('/predict/api'):** This is a POST method endpoint designed to receive environmental parameters in JSON format and return the predicted plant health status as a JSON object.

**Implementation Details:**

The Flask application is structured to handle POST requests at the **/predict/api** endpoint, where it expects a JSON object containing the environmental parameters. Upon receiving a request, the application extracts the parameters, scales them using the loaded scaler, and then passes them to the loaded model for prediction. The model's prediction is then mapped to the respective plant health category, and the result is returned as a JSON object.

**Error Handling:**

The application is equipped with error handling mechanisms to manage potential issues that might arise during the prediction process, such as incorrect data formats or missing parameters. In case of an error, the application returns a JSON object containing the error status and message.

**Integration and Testing of Plant Health Prediction API with Frontend**

The goal was to integrate the developed API with a front-end interface and conduct preliminary testing to ensure seamless interaction between the frontend and the API. The front end serves as a user-friendly interface allowing users to input environmental variables and receive plant health predictions. We can integrate the API with any frontend framework or any mobile application. Here for testing I just use HTML, CSS, JavaScript with Ajax.

**Development of Frontend Interface:**

A simple yet intuitive HTML interface was developed, allowing users to input environmental variables such as humidity, nitrogen level, ambient temperature, and presence of sunlight. The interface is designed with user-friendly input fields and a submit button to initiate the prediction. The predicted result is then displayed on the same interface, providing users with immediate feedback.

**Integration with API:**

The front-end interface is integrated with the API using AJAX, allowing asynchronous communication between the frontend and the API. Upon submitting the form, an AJAX POST request is sent to the API’s prediction endpoint with the input environmental variables in JSON format. The API processes the received data and returns the predicted plant health status, which is then displayed on the front-end interface.

**User Interaction:**

Users interact with the system by entering the environmental variables into the provided input fields and submitting the form. The system then displays the predicted plant health status on the interface, allowing users to understand the health condition of the plant based on the provided environmental variables.

**Testing and Validation:**

Preliminary testing was conducted to ensure that the frontend correctly sends the input data to the API and accurately displays the received prediction. The integration was validated by observing the seamless interaction and correct display of predictions on the front-end interface.

Here is the screenshot:

A screenshot of a medical test

Description automatically generated

**Conclusion:**

* In conclusion, considering the complexity of sensor data and the need for a highly accurate model, the Keras Neural Network model seems to be the most suitable choice for this task. It would be beneficial to continue monitoring and evaluating the model's performance as more data becomes available and make necessary adjustments to ensure sustained high performance. The development of the Plant Health Prediction API marks a significant milestone in our project, allowing for the integration of our predictive model into various applications and services. The API is structured to efficiently handle requests and provide accurate predictions, contributing to the advancement of smart farming solutions.