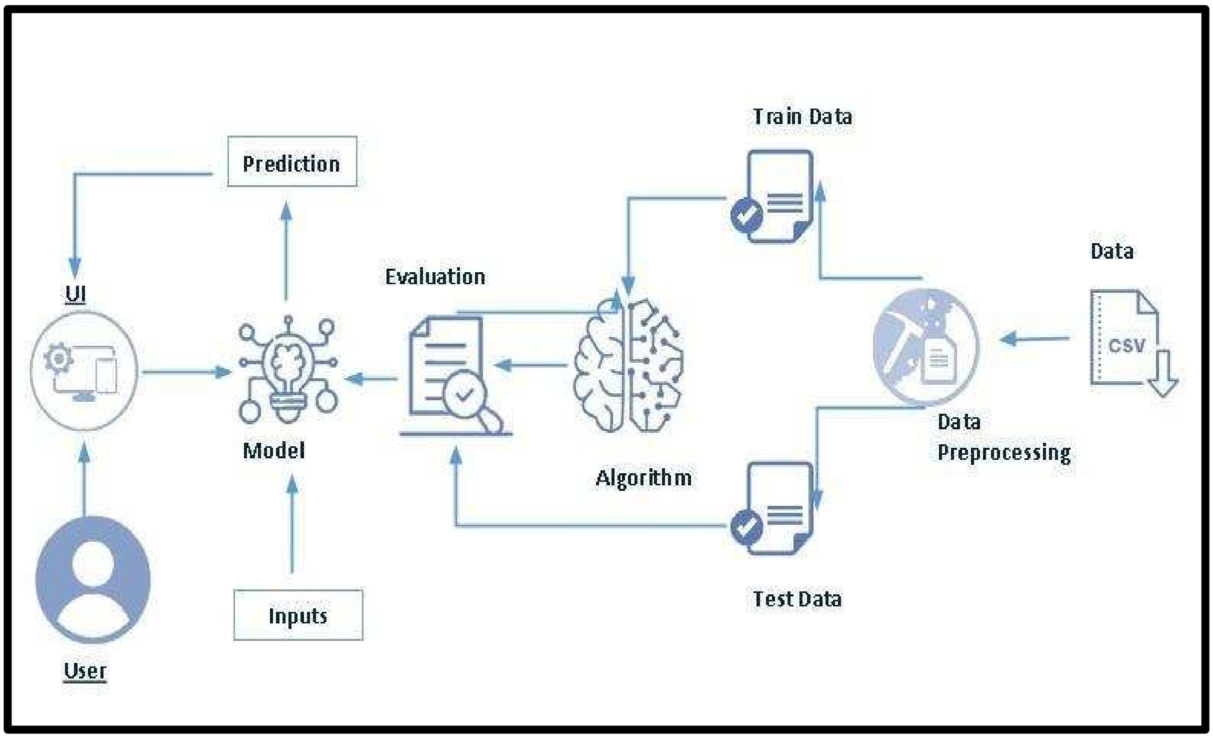
# Smart Agricultural Production Optimizing Engine Using ML

## Project Description:

- A Smart Agricultural Production Optimizing Engine Using ML is a system that uses machine learning to help farmers optimize their crop yields. The system takes into account a variety of factors, such as the type of crop, the soil conditions, the climate, and the weather forecast, To recommend the crop for the suitable soil.
- Choose the right crops to plant. The system can analyze historical data to determine which crops have historically been most successful in a particular region. This information can help farmers to make more informed decisions about what to plant.
- Optimize planting dates. The system can use weather forecasts to predict the best time to plant crops. This can help to ensure that crops are planted at the optimal time for maximum yields.
- Manage water and fertilizer use. The system can help farmers to determine the optimal amount of water and fertilizer to use for each crop. This can help to reduce costs and improve yields.
- This can help farmers to take action to To recommend the crop for the suitable soil..
- Here are some of the benefits of using a Smart Agricultural Production Optimizing Engine Using ML:

    - Increased crop yields
    - Reduced costs
    - Improved food security
    - More sustainable farming practices
    - Better decision-making
    - Increased efficiency
    - Improved profitability

## Technical Architecture:



## Pre requisites:

**To complete this project, you must required following software's, concepts and packages**

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator
  - Link : [https://youtu.be/1ra4zH2G4o0](https://youtu.be/1ra4zH2G4o0)

- **Python packages:**
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install pickle-mixin" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install Flask" and click enter.

## Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - Supervised learning: [https://www.javatpoint.com/supervised-machine-learning](https://www.javatpoint.com/supervised-machine-learning)
  - Unsupervised learning: [https://www.javatpoint.com/unsupervised-machine-learning](https://www.javatpoint.com/unsupervised-machine-learning)
  - 
  - KNN: [https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning](https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning)
  - Logistic Regression: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-

    - end-guide-to-understand-the-math-behind-logistic-regression/
    - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

## Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

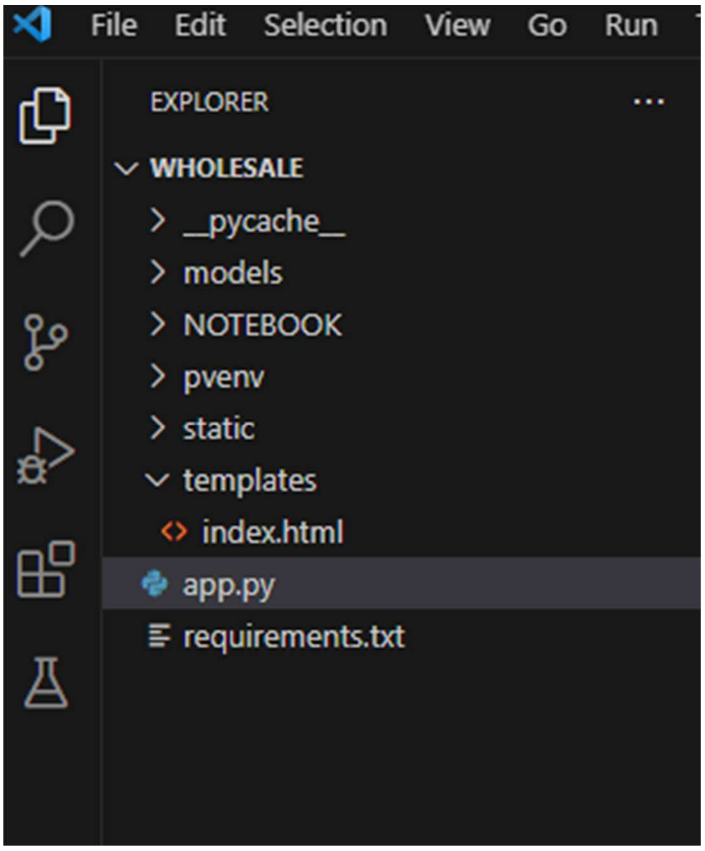
## Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
    - Collect the dataset or create the dataset
- Visualizing and analyzing data
    - Univariate analysis
    - Bivariate analysis
    - Multivariate analysis
    - Descriptive analysis
- Data pre-processing
    - Checking for null values
    - Handling outlier
    - Handling categorical data
    - Splitting data into train and test
- Model building
    - Import the model building libraries
    - Initializing the model
    - Training and testing the model
    - Evaluating performance of model
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code

## Project Structure:



## Create the Project folder which contains files as shown below

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- model.pkl is our saved model. Further we will use this model for flask integration.

## Milestone 1: Data Collection

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

**Activity 1: Download the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used Crop_recommendation.csv data. This data is downloaded from kaggle.com. Please refer the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/chitrakumari25/smart-agricultural-production-optimizing-engine/code?select=Crop_recommendation.csv

## Milestone 2: Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

**Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.**

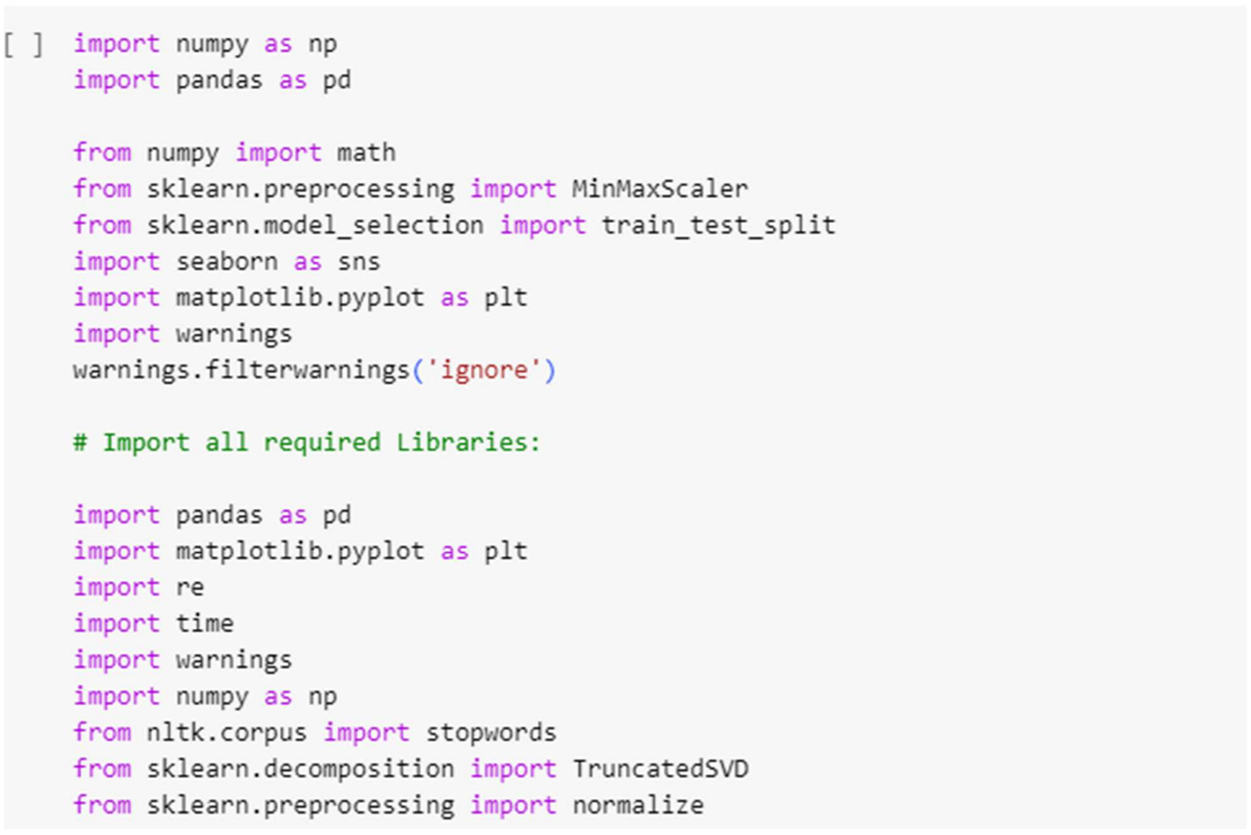
### Activity 1: Importing the libraries

Import the necessary libraries as shown in the image.

### Activity 2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.



```
[ ]  import numpy as np
     import pandas as pd

     from numpy import math
     from sklearn.preprocessing import MinMaxScaler
     from sklearn.model_selection import train_test_split
     import seaborn as sns
     import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')

     # Import all required Libraries:

     import pandas as pd
     import matplotlib.pyplot as plt
     import re
     import time
     import warnings
     import numpy as np
     from nltk.corpus import stopwords
     from sklearn.decomposition import TruncatedSVD
     from sklearn.preprocessing import normalize
```

## Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

- Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

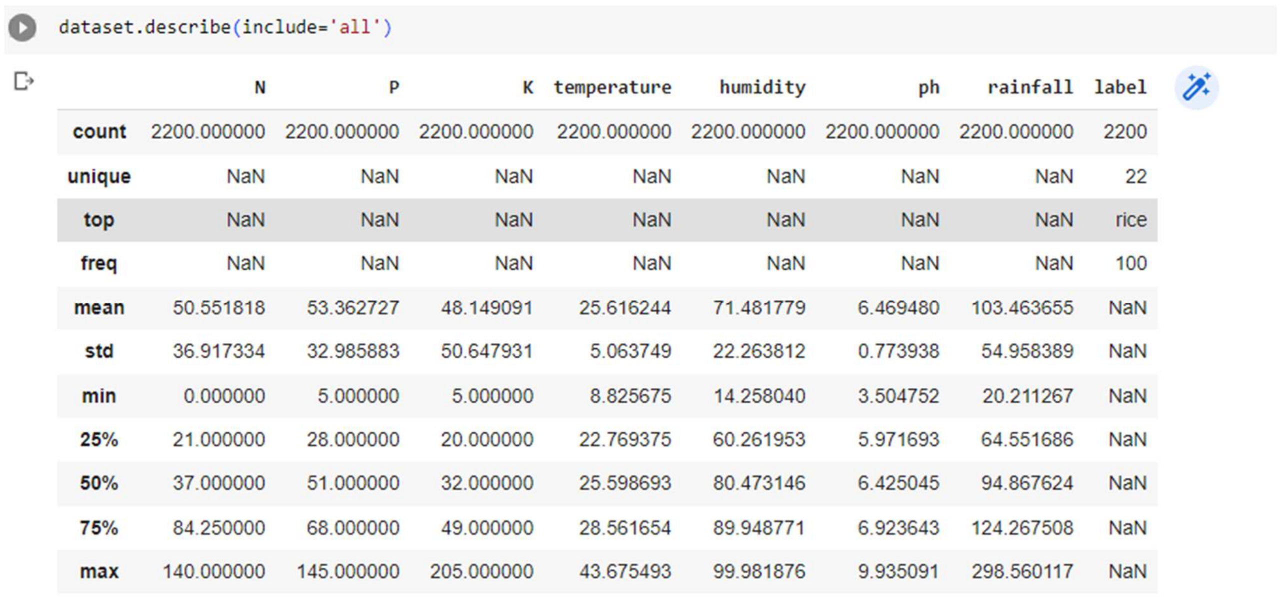
## Activity 4: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between ‘humidity’ and ‘label’ variables.

## Activity 5: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used countplot from seaborn package.

## Activity 6: Descriptive analysis



```
dataset.describe(include='all')
```

```
                  N            P            K  temperature     humidity           ph     rainfall  label
count   2200.000000  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000   2200
unique          NaN          NaN          NaN          NaN          NaN          NaN          NaN     22
top             NaN          NaN          NaN          NaN          NaN          NaN          NaN   rice
freq            NaN          NaN          NaN          NaN          NaN          NaN          NaN    100
mean      50.551818    53.362727    48.149091    25.616244    71.481779     6.469480   103.463655    NaN
std       36.917334    32.985883    50.647931     5.063749    22.263812     0.773938    54.958389    NaN
min        0.000000     5.000000     5.000000     8.825675    14.258040     3.504752    20.211267    NaN
25%       21.000000    28.000000    20.000000    22.769375    60.261953     5.971693    64.551686    NaN
50%       37.000000    51.000000    32.000000    25.598693    80.473146     6.425045    94.867624    NaN
75%       84.250000    68.000000    49.000000    28.561654    89.948771     6.923643   124.267508    NaN
max      140.000000   145.000000   205.000000    43.675493    99.981876     9.935091   298.560117    NaN
```

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

## Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

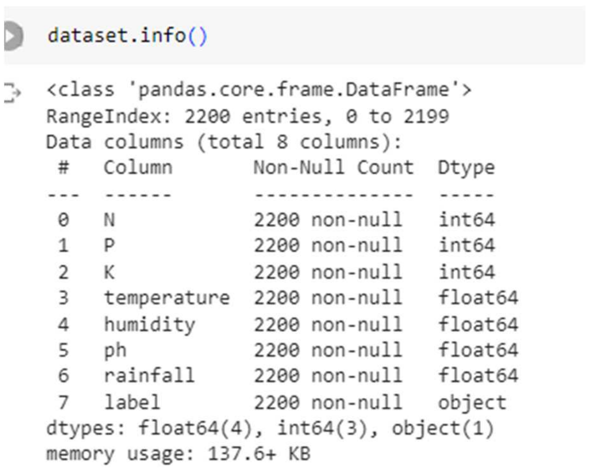
Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

### Activity 1: Checking for null values

- Let's find the shape of our dataset first, To find the shape of our data, df.shape method is used. To find the data type, df.info() function is used.

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling of missing values step.

Let's look for any outliers in the dataset



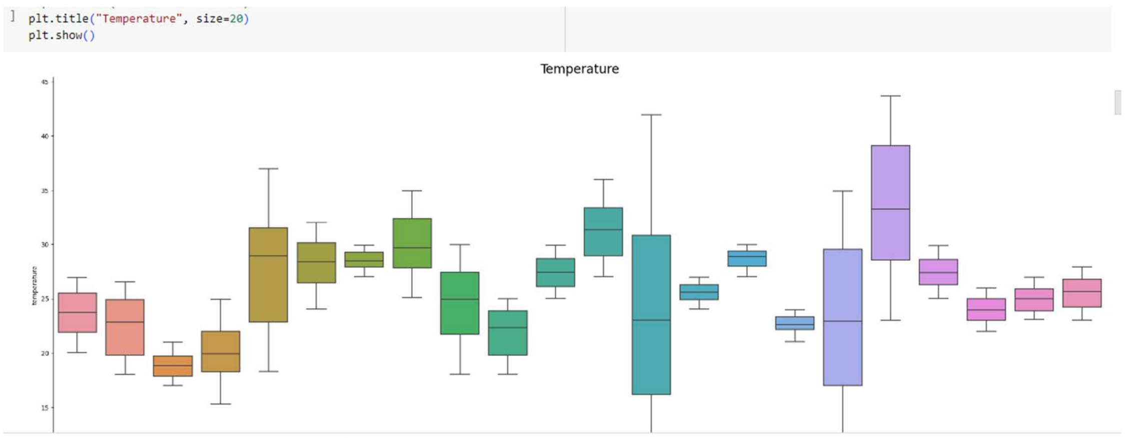
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   N            2200 non-null   int64
 1   P            2200 non-null   int64
 2   K            2200 non-null   int64
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

**Activity 2: Handling outliers**

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of all features with some mathematical formula.

- From the below diagram, we could visualize that Pottasium feature has outliers. Boxplot from seaborn library is used here.

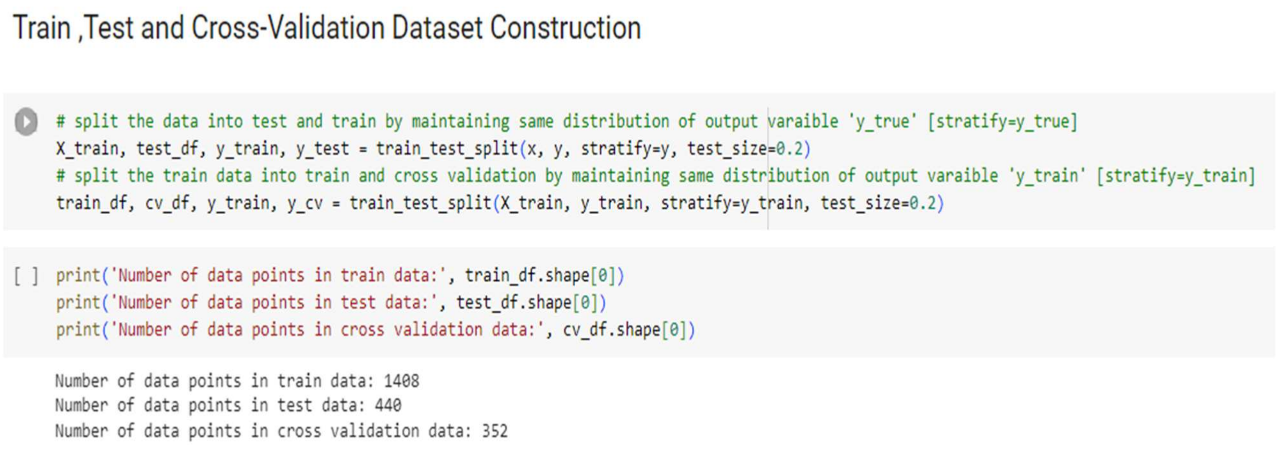


```
plt.title("Temperature", size=20)
plt.show()
```

**Activity 3: Splitting data into train and test**

Now let’s split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



Train ,Test and Cross-Validation Dataset Construction

```
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
Number of data points in train data: 1408
Number of data points in test data: 440
Number of data points in cross validation data: 352
```

## Milestone 4: Model Building

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four  classification algorithms. The best model is saved based on its performance.

Link:

https://colab.research.google.com/drive/1DTP1zoV5R8w5qMidv07mPRjxIiSUDq4A?usp=sharing

## Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

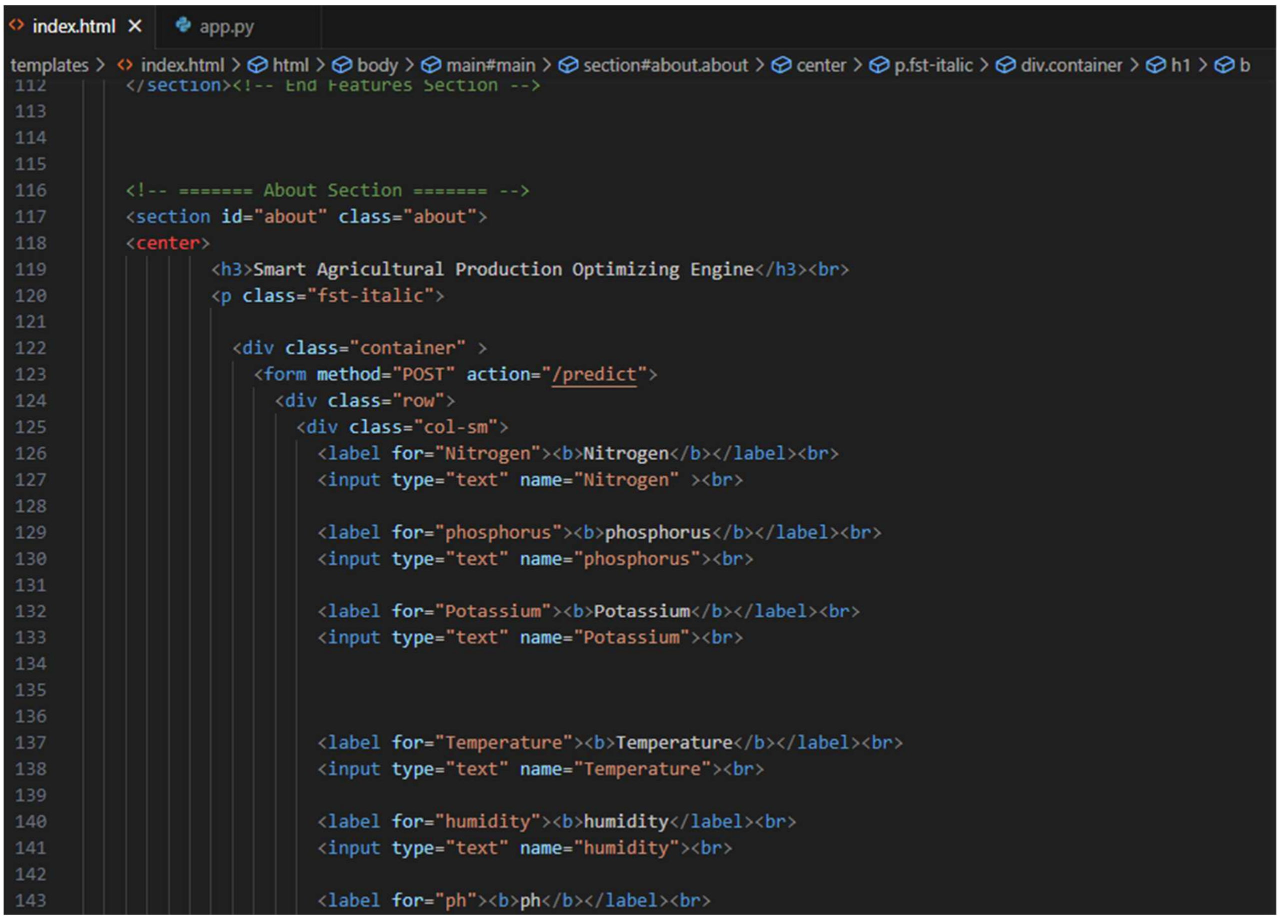
This section has the following tasks

- Building HTML Pages
- Building serverside script

### Activity1: Building Html Pages:

For this project create three HTML files namely

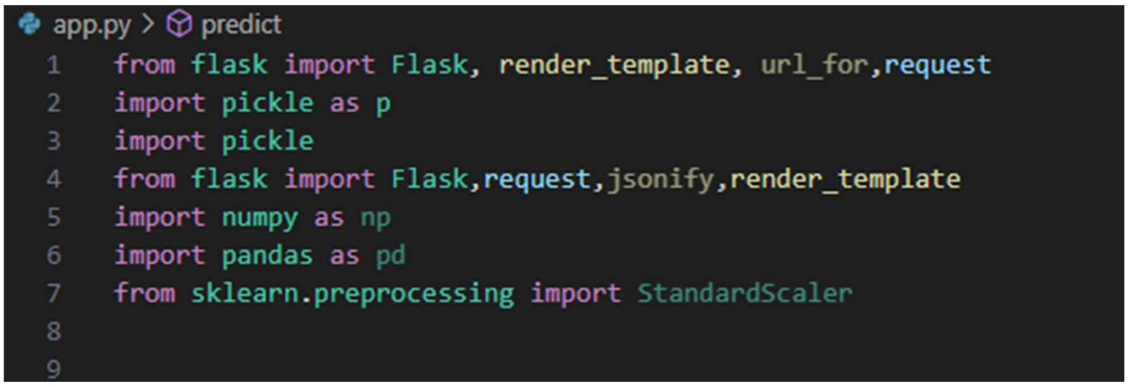
- Index.html
  and save them in templates folder.


index.html ×
app.py
templates > index.html > html > body > main#main > section#about.about > center > p.fst-italic > div.container > h1 > b

```html
    </section><!-- End Features Section -->



    <!-- ======= About Section ======= -->
    <section id="about" class="about">
    <center>
            <h3>Smart Agricultural Production Optimizing Engine</h3><br>
            <p class="fst-italic">

              <div class="container" >
                <form method="POST" action="/predict">
                  <div class="row">
                    <div class="col-sm">
                      <label for="Nitrogen"><b>Nitrogen</b></label><br>
                      <input type="text" name="Nitrogen" ><br>

                      <label for="phosphorus"><b>phosphorus</b></label><br>
                      <input type="text" name="phosphorus"><br>

                      <label for="Potassium"><b>Potassium</b></label><br>
                      <input type="text" name="Potassium"><br>



                      <label for="Temperature"><b>Temperature</b></label><br>
                      <input type="text" name="Temperature"><br>

                      <label for="humidity"><b>humidity</label><br>
                      <input type="text" name="humidity"><br>

                      <label for="ph"><b>ph</b></label><br>
```

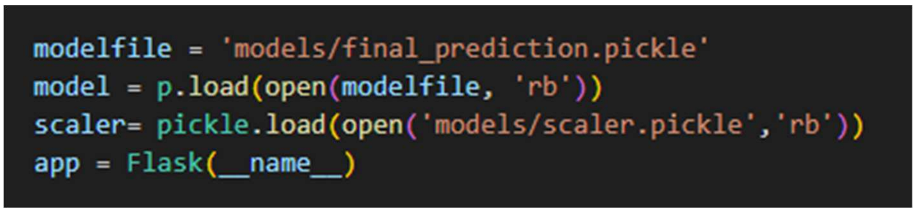
### Activity 2: Build Python code:

Import the libraries



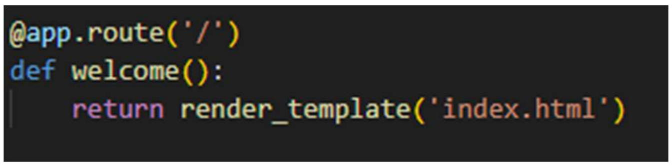
```
app.py > predict
from flask import Flask, render_template, url_for,request
import pickle as p
import pickle
from flask import Flask,request,jsonify,render_template
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler


```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.



```
modelfile = 'models/final_prediction.pickle'
model = p.load(open(modelfile, 'rb'))
scaler= pickle.load(open('models/scaler.pickle','rb'))
app = Flask(__name__)
```

Render HTML page:

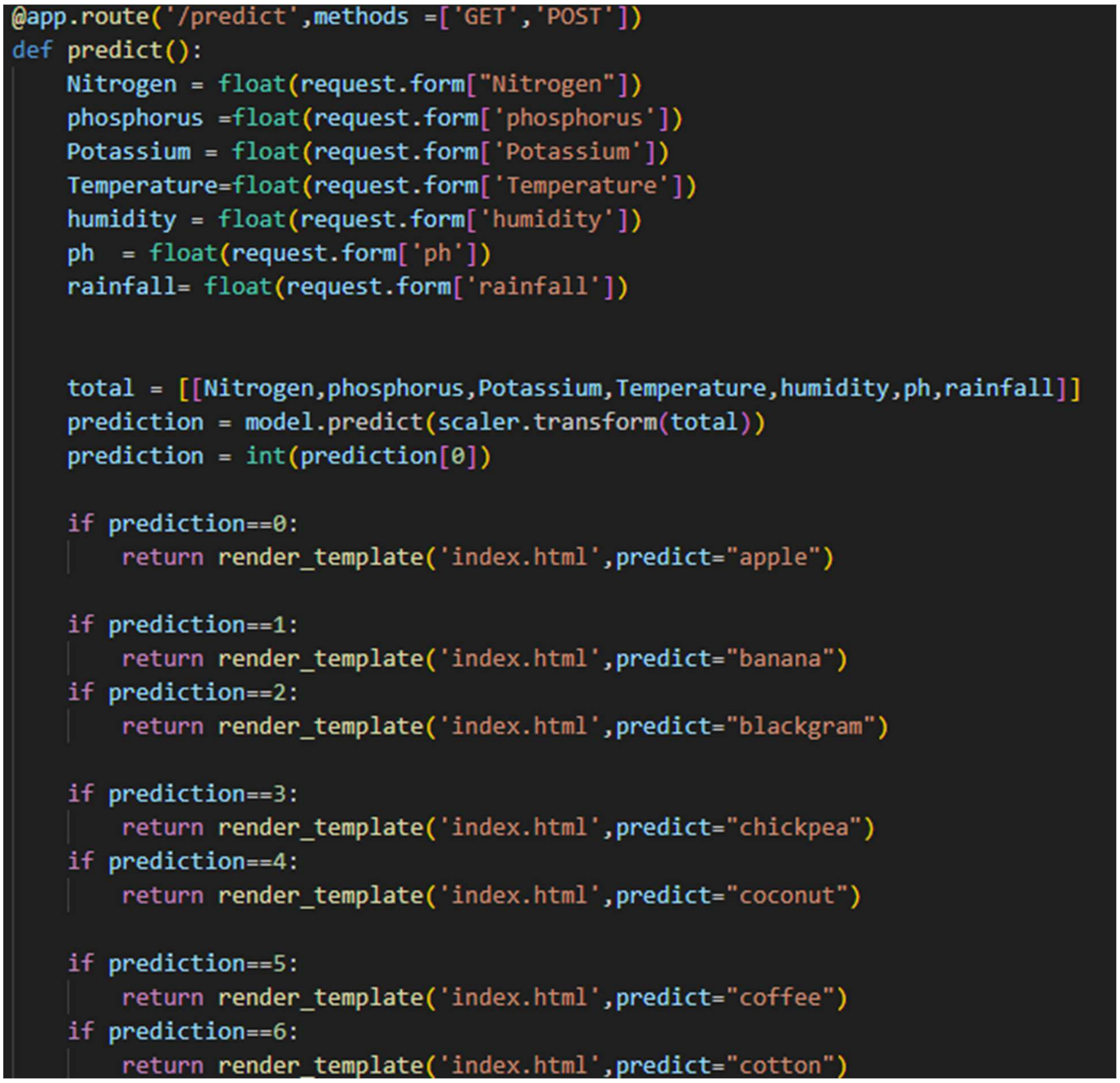


```
@app.route('/')
def welcome():
    return render_template('index.html')
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

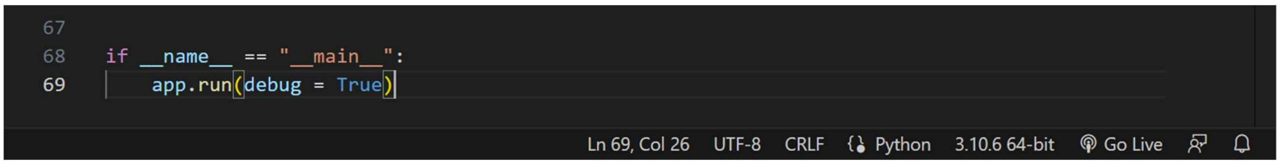
Retrieves the value from UI:



```python
@app.route('/predict',methods =['GET','POST'])
def predict():
    Nitrogen = float(request.form["Nitrogen"])
    phosphorus =float(request.form['phosphorus'])
    Potassium = float(request.form['Potassium'])
    Temperature=float(request.form['Temperature'])
    humidity = float(request.form['humidity'])
    ph  = float(request.form['ph'])
    rainfall= float(request.form['rainfall'])


    total = [[Nitrogen,phosphorus,Potassium,Temperature,humidity,ph,rainfall]]
    prediction = model.predict(scaler.transform(total))
    prediction = int(prediction[0])

    if prediction==0:
        return render_template('index.html',predict="apple")

    if prediction==1:
        return render_template('index.html',predict="banana")
    if prediction==2:
        return render_template('index.html',predict="blackgram")

    if prediction==3:
        return render_template('index.html',predict="chickpea")
    if prediction==4:
        return render_template('index.html',predict="coconut")

    if prediction==5:
        return render_template('index.html',predict="coffee")
    if prediction==6:
        return render_template('index.html',predict="cotton")
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

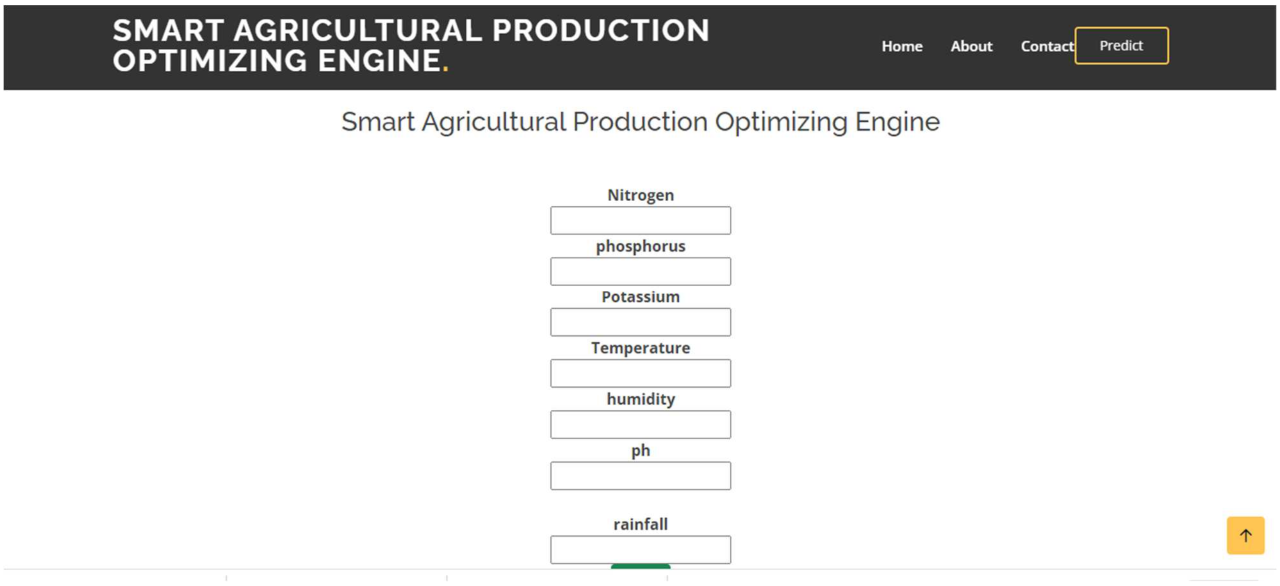


```python

if __name__ == "__main__":
    app.run(debug = True)
```

## Activity 3: Run the application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type "python app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the predict, enter the inputs, click on the submit button, and see the result/prediction on the web.

## Final Output :


SMART AGRICULTURAL PRODUCTION OPTIMIZING ENGINE.
Home
About
Contact
Predict
Smart Agricultural Production Optimizing Engine
Nitrogen
phosphorus
Potassium
Temperature
humidity
ph
rainfall