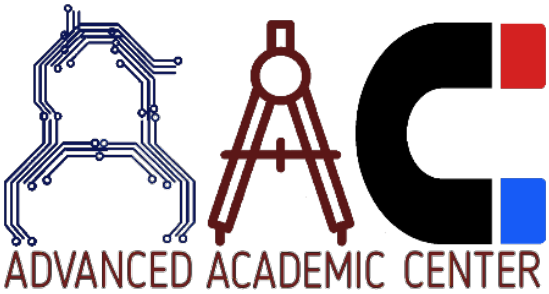


A CENTER FOR INTER-DISCIPLINARY RESEARCH

TITLE

"I-FARM"

SUPERVISED BY

PADMINI MADHIRA, SRI HARSHA

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

AUTONOMOUS

Advanced Academic Center

(A Center For Inter-Disciplinary Research)

This is to certify that the project titled

"I-FARM"

is a bonafide work carried out by the following students in partial fulfilment of the requirements for Advanced Academic Center intern, submitted to the chair, AAC during the academic year 2020-21.

NAME	ROLL NUMBER	BRANCH
BALASRI NITIN REDDY VANIPENTA	20241A6659	AIML
DEEKSHITHA GADI	20241A6615	AIML
MOHIT SURYADEVARA	20241A6657	AIML
SATHYA ABHIJNA MARISETTI	20241A6631	AIML
SREE NAGA SREEJA DOGGA	20241A6613	AIML

This work was not submitted or published earlier for any study

Dr/Ms./Mr

_____

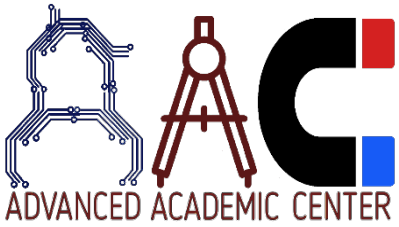
Project Supervisor

Dr.B.R.K.Reddy

Program Coordinator

Dr.Ramamurthy Suri

Associate Dean,AAC



ACKNOWLEDGEMENTS

We express our deep sense of gratitude to our respected Director, Gokaraju Rangaraju Institute of Engineering and Technology, for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we extend our appreciation to our respected Principal, for permitting us to carry out this project.

We are thankful to the Associate Dean, Advanced Academic Centre, for providing us an appropriate environment required for the project completion.

We are grateful to our project supervisor who spared valuable time to influence us with their novel insights.

We are indebted to all the above mentioned people without whom we would not have concluded the project.

INDEX

1.ABSTRACT

2.INTRODUCTION

3.YIELD PREDICTION

  3.1.PROJECT WORKFLOW

  3.2.CODE

4.PRICE PREDICTION

  4.1.PROJECT WORKFLOW

  4.2.CODE

5.SOIL HEALTH PREDICTION

  5.1.PROJECT WORKFLOW

  5.2.CODE

6.DISEASE DETECTION

  6.1.PROJECT WORKFLOW

  6.2.CODE

7.UI (USER INTERFACE)

8.GITHUB (LINK)

9.FUTURE DEVELOPMENTS

10.REFERENCES

1.ABSTRACT

Agriculture Sector is a major contributor to Indian Economy. In a country like India, which has ever increasing demand of food due to rising population, advances in agriculture sector are required to meet the needs. India's agriculture sector is under crisis for nearly two decades now. The suicidal cases are growing in numbers over the years. This roots to the lack of proper knowledge and the mundane methods adopted by farmers in their farming. Various seasonal, economic and biological patterns influence the crop production. Catastrophic changes in these patterns may lead to a great loss to the farmers. These risks can be avoided by adopting smart farming methodologies i.e., incorporating technology in the day-to-day farming. This project mainly focuses on deriving useful insights on crop-yield prediction, soil health prediction, crop disease detection and crop cost forecasting. The statistical agricultural dataset is undertaken for experimental analysis. The data is preprocessed and classified into training and testing data. Then suitable machine learning and deep learning techniques are used to produce better outcome.

2.INTRODUCTION

Agriculture is taken into account the foremost vital occupations in our country. It is the backbone of our economy and it helps in the overall development of the country. Nearly 60% of the land within the country is employed for agriculture so as to satisfy the needs of a billion individuals. Thus, the modernization of agriculture is very important and can lead the farmers of our country towards profit. Currently, India is generating negative returns. Thanks to the shortage of information and the strategies adopted by farmers in their farming. Indian farmers do not embrace technology and they work on a random basis. Numerous factors have an effect on their crops production that they're not aware of. Any changes within the weather, the economy can cause severe injury to their crops. This has resulted in a situation where farmers have low financial gain and high debts and they end up committing suicide.[1]

This project presents the concept of smart farming where agriculture is done by precisely managing data related to soil type, temperature, rainfall, and crop type field parameters in order to achieve optimized outputs at minimum disturbances to the environment. So, we have designed the system using machine learning algorithms for betterment of farmers. Our system will predict the yield of the crop based on content and weather parameters. And also, the system predicts the harvest price of the crop and predicts the crop disease and provides information about the required content and quantity of fertilizers, hence by utilizing our system farmers can predict the yield for their crop.

3.YIELD PREDICTION

Crop Yield Prediction is the methodology to predict the yield of the crops using different parameters like rainfall, temperature, area and other atmospheric conditions and parameters.

3.1.PROJECT WORKFLOW

REQUIREMENTS

The system requires inputs such as state, year, crop name, area, rainfall(mm) from the user. Based on the given parameters the system will predict the yield (Quintals) for the crop.

DEFINING

Predicting Yield of the Crop Using Machine Learning Algorithm is based on the central theme that certain factors like weather conditions, soil parameters and the historic details of the crop has an effect on the yield of the crop in the present. So, it is important to take these factors or the parameters into consideration and predict the yield of the crop planted. Certain Machine Learning algorithms such as Random Forest is taken into account for developing the models of the data obtained from the past historical yield of the crop and reflecting them in predicting the yield of the crop planted in the present.[2]

DATASET

The dataset is collected from Kaggle.com and then analyzed and prepared.

1) https://www.kaggle.com/chinmaynagesh/crop-yield-per-state-and-rainfall-data-of-india

```
df.head()
```

	State	Year	Season	Crop	Area	Rainfall	Production
0	Andaman and Nicobar Islands	2000	Kharif	Rice	102.0	2763.2	321.0
1	Andaman and Nicobar Islands	2000	Whole Year	Banana	176.0	2763.2	641.0
2	Andaman and Nicobar Islands	2000	Whole Year	Dry ginger	36.0	2763.2	100.0
3	Andaman and Nicobar Islands	2000	Whole Year	Sugarcane	1.0	2763.2	2.0
4	Andaman and Nicobar Islands	2000	Whole Year	Sweet potato	5.0	2763.2	15.0

Dataset that has been used contains the data of 33 various crops.

```
df_original.Crop.value_counts()
```

```
Rice                        5882
Maize                       4827
Moong(Green Gram)           3532
Urad                        3171
Sesamum                     2808
Wheat                       2584
Sugarcane                   2576
Rapeseed &Mustard           2450
Potato                      2436
Groundnut                   2346
Ragi                        2183
Arhar/Tur                   2174
Horse-gram                  1987
Gram                        1920
Dry chillies                1644
Onion                       1637
Small millets               1579
Turmeric                    1563
Sunflower                   1425
Dry ginger                  1387
Masoor                      1363
Sweet potato                1334
Peas & beans (Pulses)       1331
Barley                      1258
Banana                      1164
Linseed                     1132
Coriander                   1123
Garlic                      1072
Jowar                       1070
Bajra                        930
Khesari                      838
Cotton(lint)                 829
Castor seed                  805
```

INPUTS

1)State

2)Year

3)Season (Kharif, Rabi, Whole Year)

4)Crop Name

5)Rainfall(mm)

6)Area (Acres)

OUTPUT

1)Yield of the Crop (Quintals)

3.2.CODE

DATA PREPROCESSING

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.[3]

The dataset obtained from Kaggle consists of data belonging to 86 different crops, so to get accurate predictions we deleted a few crops which has insufficient data.

removed crops whose Frequency is less than 805 lets get a list

```
last_crops = [x for x in df_original.Crop.value_counts().sort_values(ascending = False).tail(50).index]
```

```
print(last_crops)
```

```
['Sannhamp', 'Mesta', 'Tapioca', 'Cashewnut', 'Tobacco', 'Coconut', 'Soyabean', 'Arecanut', 'Jute', 'Black pepper', 'Niger seed', 'Safflo
wer', 'Oilseeds total', 'Other Cereals & Millets', 'Cardamom', 'other oilseeds', 'Pulses total', 'Guar seed', 'Mango', 'Total foodgrain',
'Jack Fruit', 'Ginger', 'Drum Stick', 'Papaya', 'Pineapple', 'Pome Fruit', 'Brinjal', 'Bhindi', 'Tomato', 'Citrus Fruit', 'Korra', 'Mot
h', 'Varagu', 'Orange', 'Samai', 'Blackgram', 'Rubber', 'Other Vegetables', 'Grapes', 'Cabbage', 'Cond-spcs other', 'Tea', 'Cashewnut Ra
w', 'Lentil', 'Coffee', 'Turnip', 'Beans & Mutter(Vegetable)', 'Redish', 'Carrot', 'other misc. pulses']
```

```
type(last_crops)
```

```
list
```

```
for label in last_crops:
  df_original.drop(df_original.index[(df_original["Crop"] == label)], axis=0, inplace=True)
```

```
type(df_original)
```

```
pandas.core.frame.DataFrame
```

Hence the final dataframe consists of data belonging to 33 different crops.

```
df_original.Crop.value_counts()
```

```
Rice                    5882
Maize                   4827
Moong(Green Gram)       3532
Urad                    3171
Sesamum                 2808
Wheat                   2584
Sugarcane               2576
Rapeseed &Mustard       2450
Potato                  2436
Groundnut               2346
Ragi                    2183
Arhar/Tur               2174
Horse-gram              1987
Gram                    1920
Dry chillies            1644
Onion                   1637
Small millets           1579
Turmeric                1563
Sunflower               1425
Dry ginger              1387
Masoor                  1363
Sweet potato            1334
Peas & beans (Pulses)   1331
Barley                  1258
Banana                  1164
Linseed                 1132
Coriander               1123
Other Kharif pulses     1111
Garlic                  1072
Jowar                   1070
Bajra                    930
Other  Rabi pulses       863
Khesari                  838
Cotton(lint)             829
Castor seed              805
Name: Crop, dtype: int64
```

SPLITTING DATASET INTO TRAIN AND TEST SET

This step includes training and testing of the input data. The stacked information is isolated into two sets, such as preparing and testing the data. Training set is mapped with the training set and during the training phase data is to be testing after learning from previous observations. The final data is formed and is processed by machine learning module.[4]

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

ALGORITHM

Random Forest is a binary tree-based machine-learning methodology. We use this algorithm to predict yields of varied crops. RF develops many decision trees based on a random selection of data and variables. More trees result in a more robust prediction. Random forest handles the missing values and handles the accuracy for it. It handles the dataset with higher dimensionality.[1]

```
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(random_state=0, n_estimators=50)
pipe = make_pipeline(ct, regr)
```

Using K-Fold cross validation

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(pipe, X, y, cv=cv)
```

```
array([0.94672771, 0.95450651, 0.94540092, 0.91165769, 0.97142466])
```

ACCURACY

After training the model, 98.08% accuracy has been achieved.

```
pipe.score(X_test, y_test)
```

```
0.980862694058047 2
```

TESTING

PREDICTIONS

Here, are a few predictions.

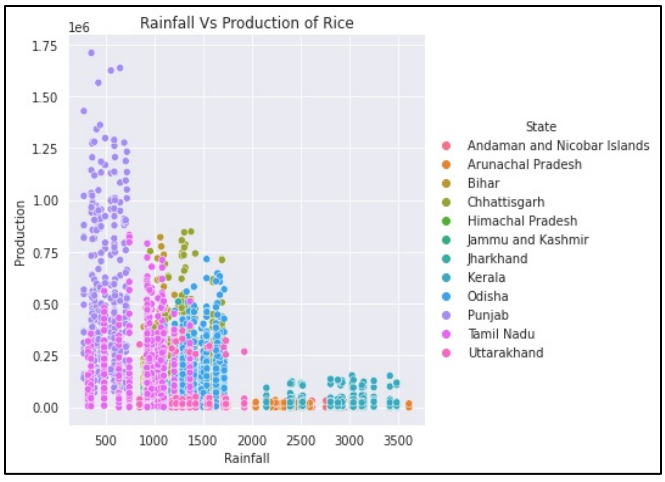
```
pipe.predict([['Chhatisgarh',2011,'Kharif','Rice', '177115','1302.7']])[0]
```

```
379223.12
```

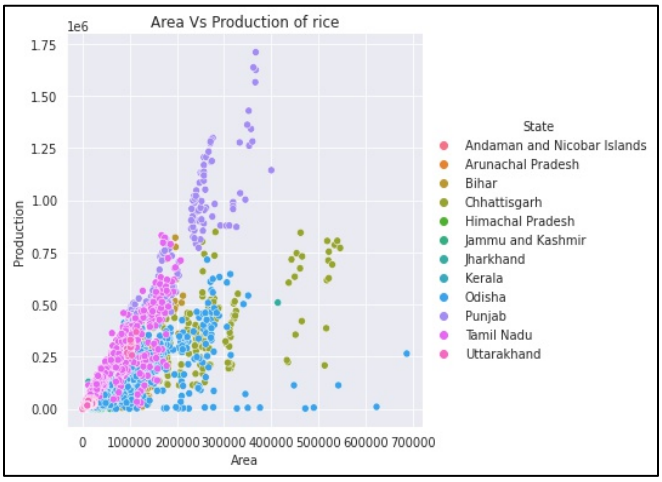
```
pipe.predict([['Andaman and Nicobar Islands', 2006, 'Whole Year', 'Banana', 198, 2404.7]])[0]
```

```
1098.5424
```

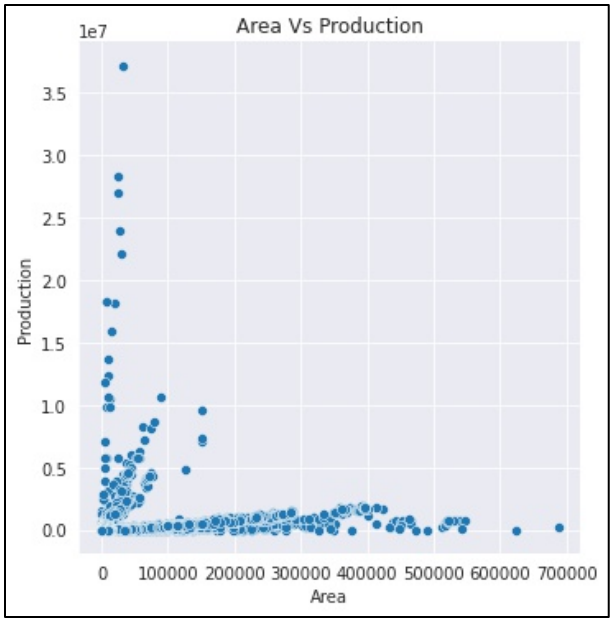
GRAPHS



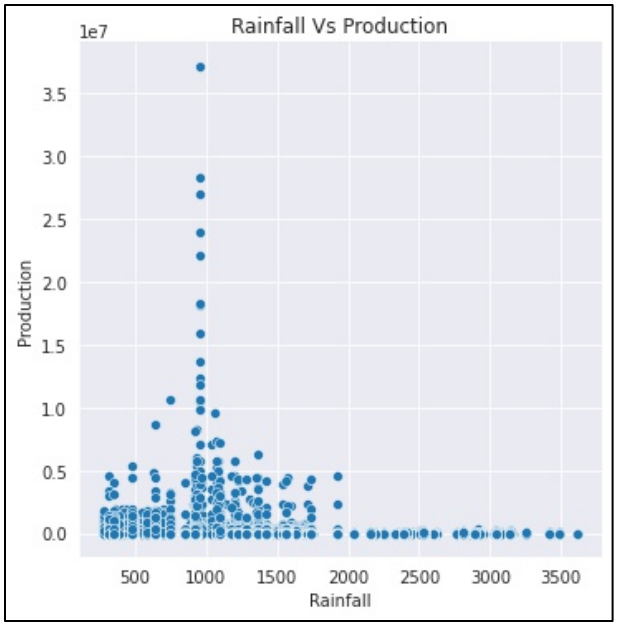
Map showing the relation between rainfall and Production of rice for the data that we have used.



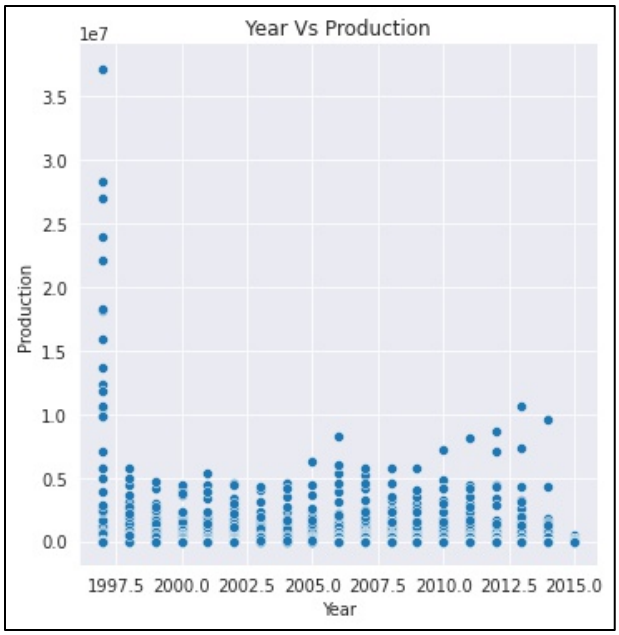
Map showing the relation between Area and Production of rice for the data that we have used.



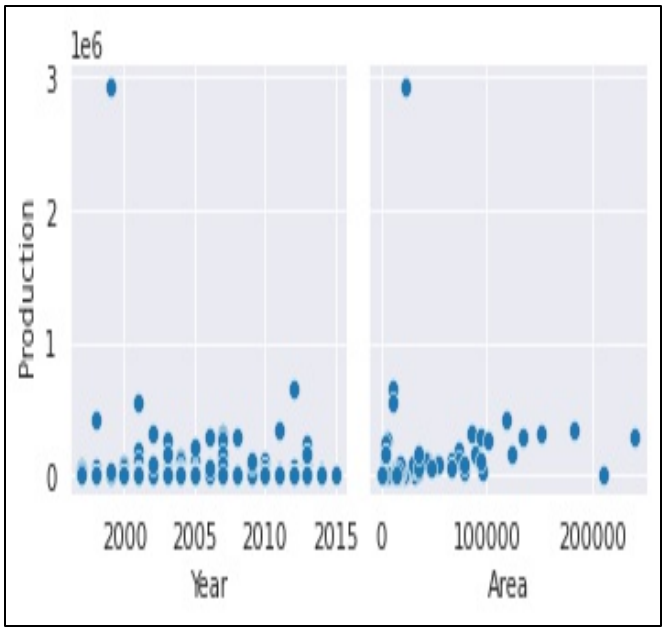
Map showing the relation between Area and Production for the data that we have used.



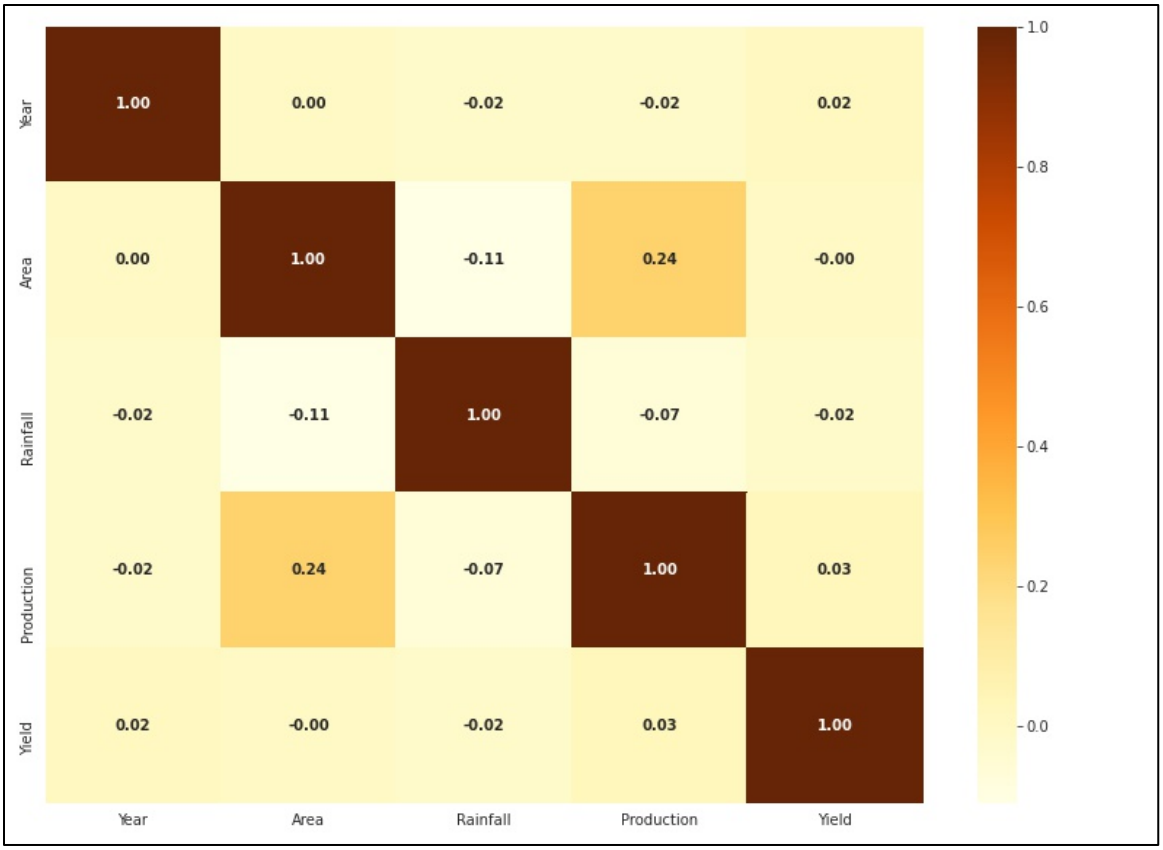
Map showing the relation between Rainfall and Production for the data that we have used.



Map showing the relation between Year and Rainfall for the data that we have used.



Map providing information on how Production is dependent on Area and Year for the data that we have used.



Heat map obtained for the data we have used.

4.PRICE PREDICTION

Crop Price Prediction System Using Machine Learning Algorithm revolves around the major issue of agriculture- Harvest Price of the crop. This project develops a system which accurately predicts the price of the crop.

4.1.PROJECT WORKFLOW

REQUIREMENTS

The system requires inputs such as yield of the crop and year from the user. Based on the given parameters the system will predict the price of the crop.

DEFINING

Price Prediction, nowadays, has become a very important agricultural problem. The aim of this project is to predict the crop price for the next rotation. This provides the farmer with an insight of what the future price of the crop that he is going to harvest.[2]

DATASET

The dataset is collected from icrisat.com and then analyzed and prepared. The dataset used in this project consists of different types of crops, various states, year and the price of the crop (Rs per quintal)

1) http://data.icrisat.org/dld/src/crops.html

```
df.head()
```

	State	Crop	Year	Cost
0	Chhattisgarh	rice	1967	110.0
1	Chhattisgarh	rice	1968	98.8
2	Chhattisgarh	rice	1969	94.0
3	Chhattisgarh	rice	1970	95.4
4	Chhattisgarh	rice	1971	95.9

Dataset that has been used contains the data of 16 different crops.

```
unique_crops = df.Crop.unique()
print(np.sort(unique_crops))
```

```
['barley' 'castor' 'chickpea' 'cotton' 'finger millet' 'groundnut'
 'linseed' 'maize' 'pearl millet' 'pigeonpea' 'rape and mustard' 'rice'
 'seasmum' 'sorghum' 'sugarcane' 'wheat']
```

INPUTS

1)State

2)Crop Name

3)Year

OUTPUT

1)Price of the crop (Rs per quintal)

4.2.CODE

ALGORITHM

To predict the price of the crop we use a machine learning algorithm known as Random Forest Regressor.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline

reg = RandomForestRegressor(n_estimators=50, random_state=0)

pipe = make_pipeline(ct, reg)
pipe.fit(X_train, y_train)
```

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(pipe, X, y, cv=cv)
```

```
array([0.94099483, 0.94286816, 0.94205089, 0.94349153, 0.93931069])
```

ACCURACY

After training the model, 94.37% accuracy has been achieved.

```
pipe.score(X_test, y_test)
```

```
0.9437391822374935
```

TESTING

PREDICTIONS

Here, are a few predictions.

```
pipe.predict([['Maharashtra', 'maize', 2016]])[0]
```

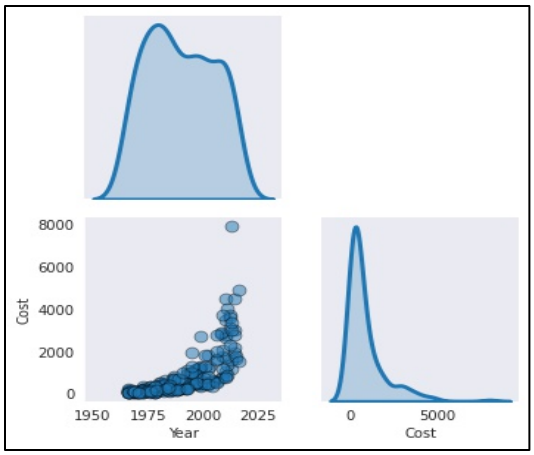
```
1393.5530886463325
```

```
pipe.predict([['Madhya Pradesh', 'rice', 2014]])[0]
```

```
2589.8950370887687
```

GRAPHS

Heat map obtained for the data that we have used.



Correlation map showing relation between Cost and year for the data that we have used.

5.SOIL HEALTH PREDICTION

Fertilizers are being used to increase crop productivity, but for producing more crop, nutrient level of soil and crop monitoring is more important. This is one of the main factors for the production of rich and good quality crops and also helps the farmers to determine the type of crops to be planted on his land.

5.1.PROJECT WORKFLOW

REQUIREMENTS

Need to predict N(nitrogen), P(phosphorus), K(potassium) values of the soil based on the various inputs given by the user.

DEFINING

Machine learning algorithms are used in predicting fertiliser requisites using historical data as input. Machine learning has Intelligence built in it. It learns from its previous experience and improves itself to predict the results more accurately. They help to increase the productivity [5][6][7]

DATASET

	label	temperature	humidity	rainfall	ph	N	P	K
0	rice	20.879744	82.002744	202.935536	6.502985	90	42	43
1	rice	21.770462	80.319644	226.655537	7.038096	85	58	41
2	rice	23.004459	82.320763	263.964248	7.840207	60	55	44
3	rice	26.491096	80.158363	242.864034	6.980401	74	35	40
4	rice	20.130175	81.604873	262.717340	7.628473	78	42	42
...	...	...	...	...	...	...	...	...
2195	coffee	26.774637	66.413269	177.774507	6.780064	107	34	32
2196	coffee	27.417112	56.636362	127.924610	6.086922	99	15	27
2197	coffee	24.131797	67.225123	173.322839	6.362608	118	33	30
2198	coffee	26.272418	52.127394	127.175293	6.758793	117	32	34
2199	coffee	23.603016	60.396475	140.937041	6.779833	104	18	30

2200 rows × 8 columns

https://www.kaggle.com/atharvaingle/crop-recommendation-dataset

Dataset that has been used contains the data of 22 various crops.

jute	100
kidneybeans	100
maize	100
rice	100
pomegranate	100
watermelon	100
mungbean	100
grapes	100
lentil	100
coffee	100
orange	100
blackgram	100
mango	100
chickpea	100
mothbeans	100
cotton	100
banana	100
pigeonpeas	100
muskmelon	100
coconut	100
papaya	100
apple	100

INPUTS

1)Crop Name

2)Temperature

3)Humidity

4)Rainfall

5)Ph

OUTPUTS

1)N (nitrogen of soil)

2)P (phosphorus of soil)

3)K (potassium of soil)

UNITS

No units. The output values will be the ratio of N, P, K to be maintained.

5.2.CODE

ALGORITHM

Random Forest Regressor has been used to train the model. A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(random_state=0, n_estimators=100)
pipe = make_pipeline(ct, regr)
```

Using K-Fold cross validation

```
from sklearn.model_selection import ShuffleSplit,cross_val_score

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(pipe, X, y, cv=cv)
```

```
array([0.9396287 , 0.94153699, 0.94015104, 0.93224548, 0.93697137])
```

ACCURACY

After training the model,93% accuracy has been achieved.

```
pipe.score(X_test, y_test)
```

```
0.9327069118695589
```

TESTING

PREDICTIONS

After training the model, we have predicted some results.

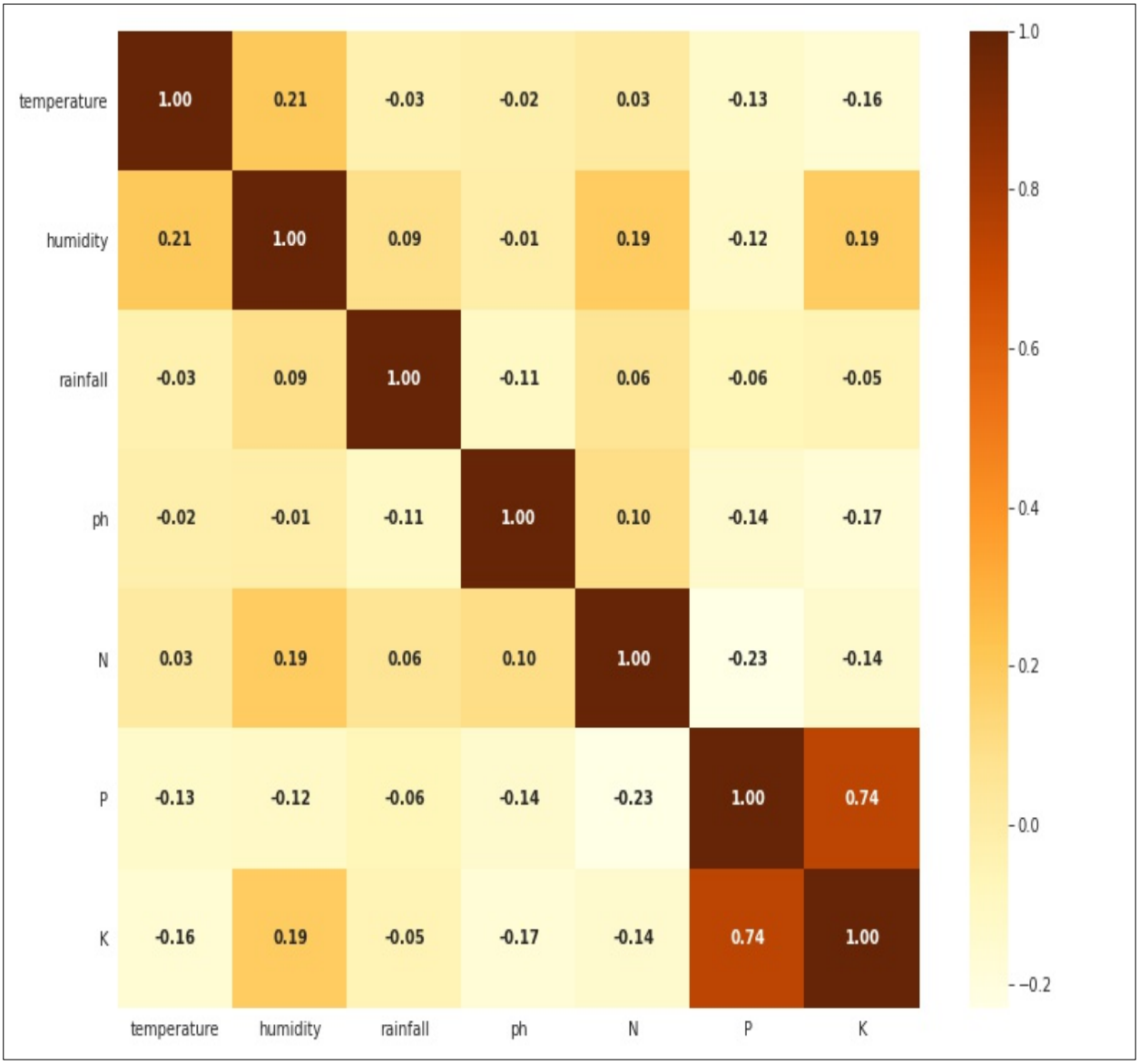
```
pipe.predict([['rice',20.87, 82, 202.9, 6.5]])
```

```
array([[88.03, 43.37, 42.24]])
```

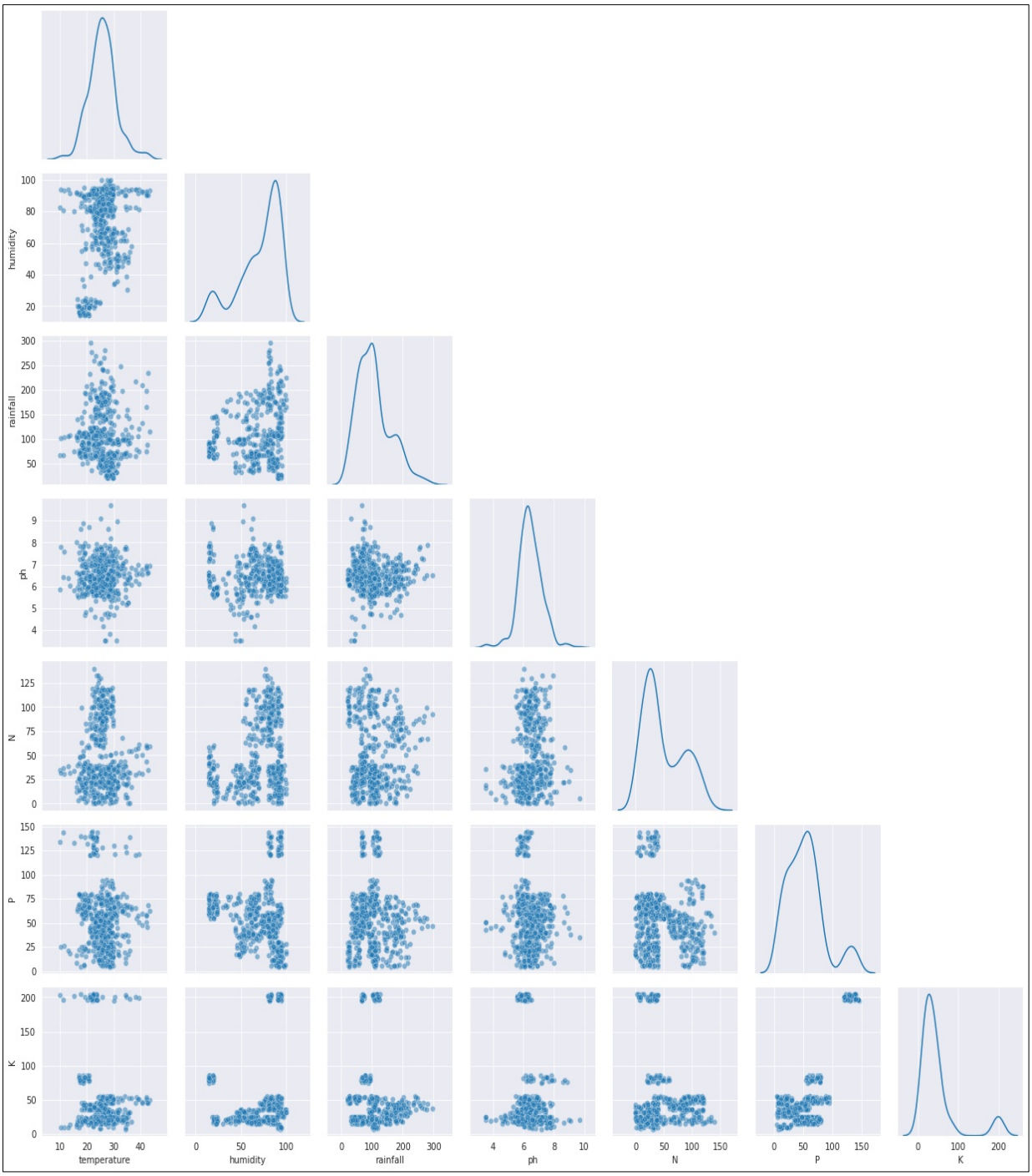
```
pipe.predict([['maize',19, 69, 80.72, 6.74]])
```

```
array([[73.9 , 44.73, 20.08]])
```

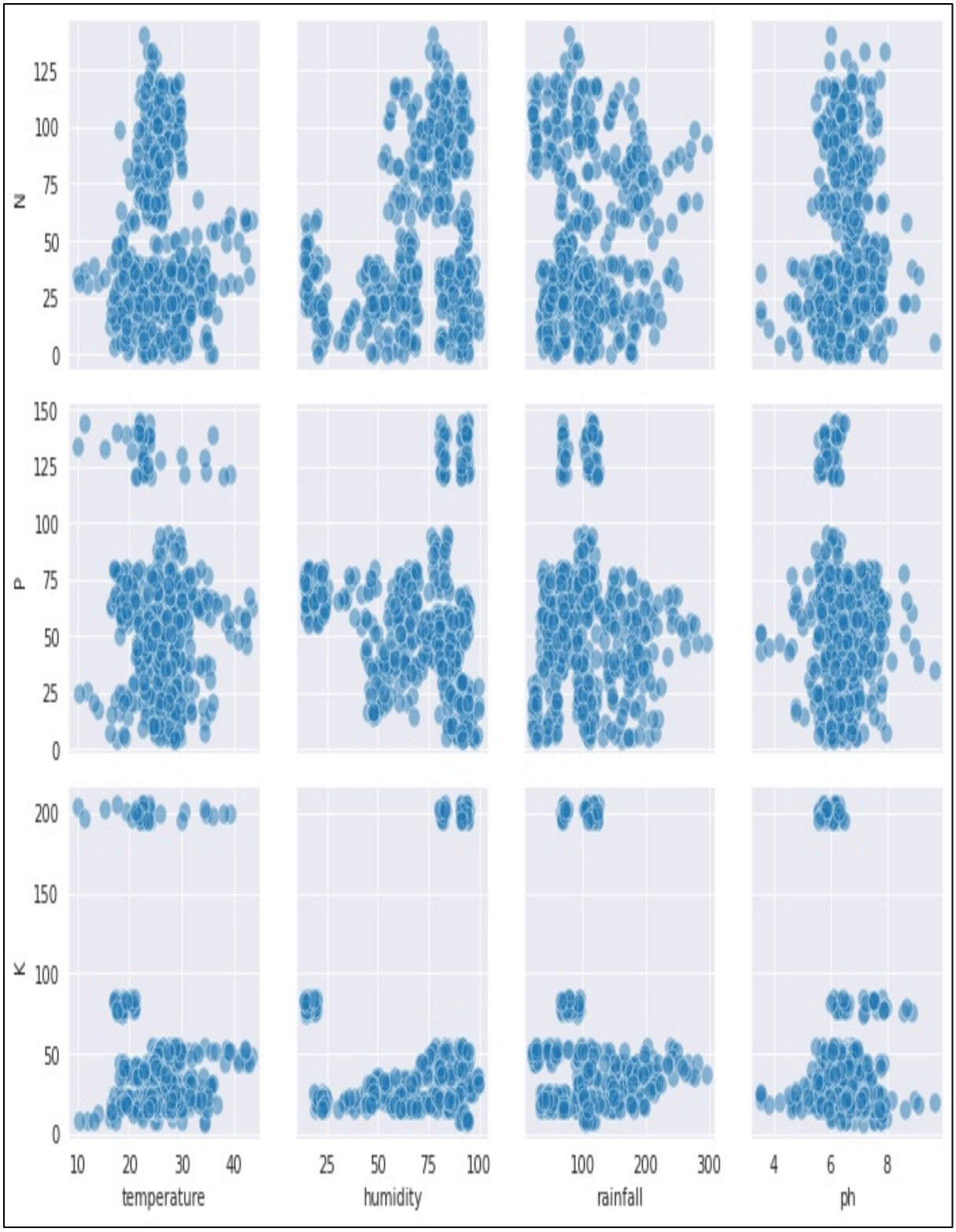
GRAPHS



Heat map for the data that we have used.



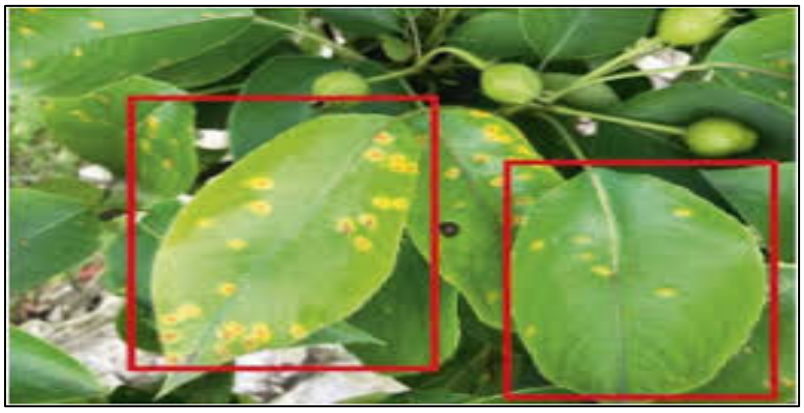
Correlation maps among the N, P, K, Ph, rainfall and humidity from the data we have used.



Maps giving the relation between how N, P, K values are dependent on temperature, humidity, rainfall.

6.PLANT DISEASE DETECTION

Crop diseases are a major threat to food security, but their rapid identification remains difficult in many parts of the world due to the lack of the necessary infrastructure. Plant diseases are not only a threat to food security at the global scale, but can also have disastrous consequences for smallholder farmers whose livelihoods depend on healthy crops.



6.1.PROJECT WORKFLOW

REQUIREMENTS

Name of the disease to be predicted when the farmer uploads the image of leaf of the plant. It should also need to distinguish between healthy and diseased crop.

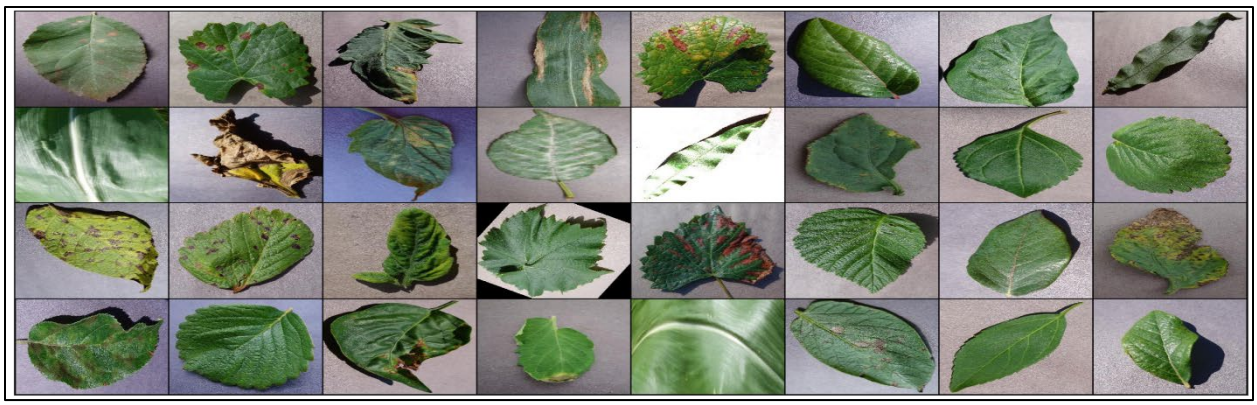
DEFINING

Deep Learning is used in-order to predict the disease of the plant. The combined factors of widespread smartphone penetration, HD cameras, and high-performance processors in mobile devices lead to a situation where disease diagnosis based on automated image recognition.[8][9]

DATASET

The dataset contains around 87.9k leaf images of 14 healthy and diseased crops which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure.

https://www.kaggle.com/vipoooool/new-plant-diseases-dataset



INPUT

1)Image of the diseased leaf.

OUTPUT

1)Name of the disease (if any identified)

2)Otherwise gives the label as healthy crop.

6.2.CODE

ALGORITHM

1)Importing required modules.

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        from PIL import Image
        import os

        import torch
        import torch.nn as nn
        from torch.utils.data import DataLoader
        import torch.nn.functional as F
        import torchvision.transforms as transforms
        from torchvision.utils import make_grid
        from torchvision.datasets import ImageFolder
        from torchsummary import summary
```

2)About Dataset and Uploading the data.

About Dataset

The dataset contains the images of both healthy and diseases of plants, the motive is to find wether plant is healthy or it is infected.

Folder structure:

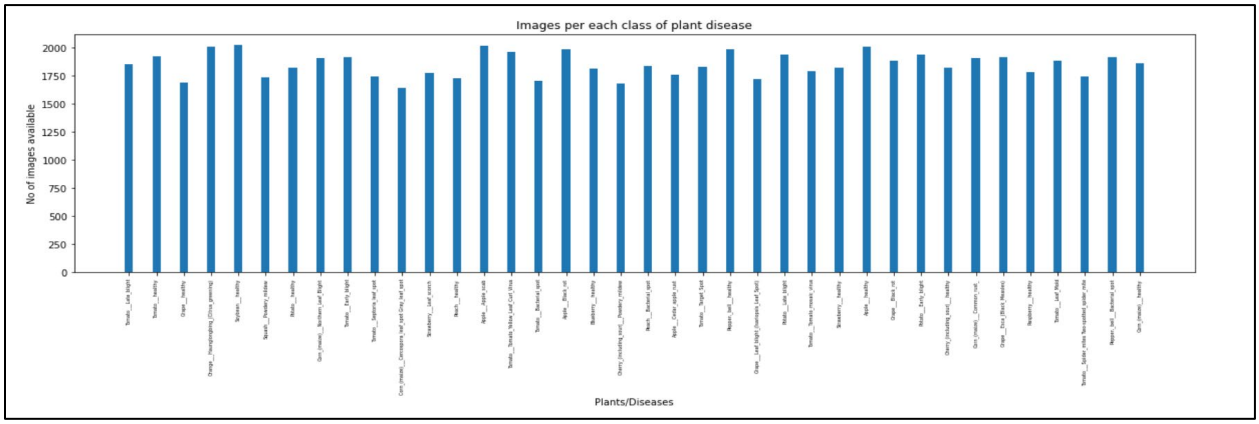
1. For Disease plants : name of plant___disease
2. For health plants : name of plant___healthy

It contains Train and valid folders

```
[4]: train_path = '../Datsets/Imgs/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train'
     valid_path = '../Datsets/Imgs/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/valid'

     total_classes = os.listdir(train_path)
```

3)Print the required number of images for each disease and also unique disease.



We can see that the data-set is almost balanced for all classes. So, we can proceed

4)Data preparation for training.

Prepeartion of data

```
In [14]: # datasets for validation and training
         train_ds = ImageFolder(train_path, transform=transforms.ToTensor())
         valid_ds = ImageFolder(valid_path, transform=transforms.ToTensor())
```

5)Defining image shape. after loading the data, we need to transform the pixel value of each image (0 -255) to 0-1.as neural networks works quite good with normalized data.

Distribution

```
In [15]: img, label = train_ds[0]
         print(img.shape, label)
```

```
torch.Size([3, 256, 256]) 0
```

image is of size 256x256 is size of the image and 3 is the number of channels (RGB)

We can see that the shape (3,256,256) of the image where 3 is the number of channels (RGB)

6)Setting the seed value and the batch size.

```
In [19]: random_seed = 8
         torch.manual_seed(random_seed)

         batch_size = 32

         # DataLoaders for training and validation
         train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=3, pin_memory=True)
         valid_dl = DataLoader(valid_ds, batch_size, num_workers=3, pin_memory=True)
```

Batch size is the total number of images given as input at once in forward propagation of the CNN. which means it defines the number of samples that will be propagated through the network.

7)Data loaders training and validation.

- Data loaders is a sub class which comes from torch. utils
- It helps in loading large and memory consuming datasets.
- setting shuffle=true shuffles the data-set
- num_workers denote the number of processes that generates batches in parallel.


```
In [19]: random_seed = 8
         torch.manual_seed(random_seed)

         batch_size = 32

         # DataLoaders for training and validation
         train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=3, pin_memory=True)
         valid_dl = DataLoader(valid_ds, batch_size, num_workers=3, pin_memory=True)
```


8)Modelling

- Note: It is advisable to use GPU instead of CPU.
- Some helper functions.

Checking Device

In [22]:
```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

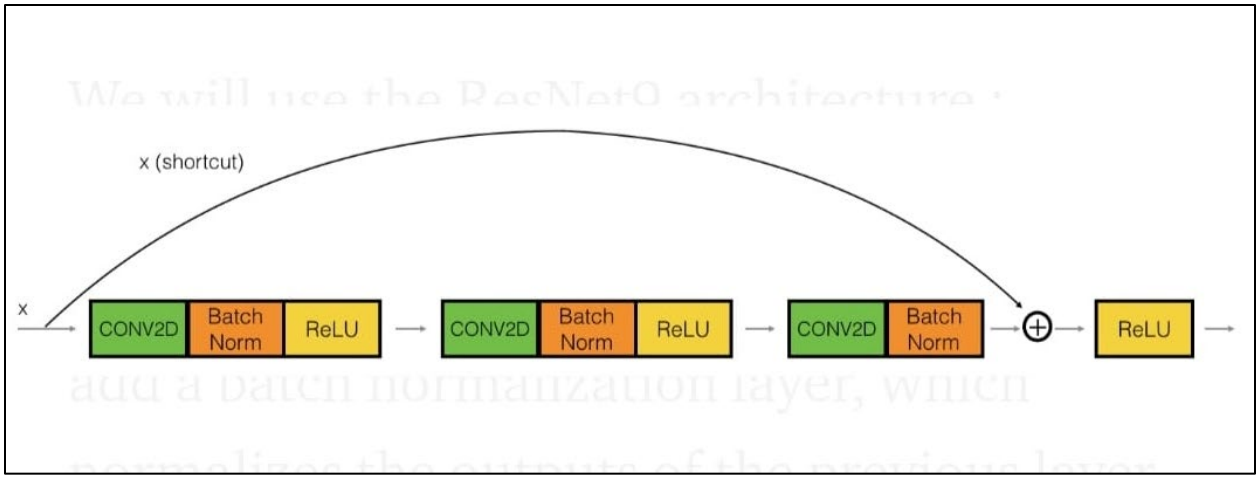
9)Moving data into GPU.

We can now wrap our training and validation data loaders using DeviceDataLoader for automatically transferring batches of data to the GPU (if available).

In [24]:
```
train_ddl = DeviceDataLoader(train_dl, device)
valid_ddl = DeviceDataLoader(valid_dl, device)
```

10) Building the architecture. we are going to use transfer learning methods ResNet, which has been one of the major breakthroughs in computer vision since they were introduced in 2015.

In ResNet unlike in traditional neural network each layer feeds into next layer .We use a network with residual blocks.



11)Residual block code implementation.

```
In [26]: class SimpleResidualBlock(nn.Module):
             def __init__(self):
                 super().__init__()
                 self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
                 self.relu1 = nn.ReLU()
                 self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
                 self.relu2 = nn.ReLU()

             def forward(self, x):
                 out = self.conv1(x)
                 out = self.relu1(out)
                 out = self.conv2(out)
                 return self.relu2(out) + x # ReLU can be applied before or after adding the input
```

12)Then we define our image classification base whose functions are

Training step: To figure out how "wrong" the model is going after training or validation step, we are using this function other than just an accuracy metric that is likely not going to be differentiable this would mean that the gradient can't be determined, which is necessary for the model to improve during training.

Validation step: Because an accuracy metric can't be used while training the model, doesn't mean it shouldn't be implemented! Accuracy in this case would be measured by a threshold, and counted if the difference between the model's prediction and the actual label is lower than that threshold.

Validation epoch end: We want to track the validation losses/accuracies and train losses after each epoch, and every time we do so we have to make sure the gradient is not being tracked.

Epoch end: We also want to print validation losses/accuracies, train losses and learning rate too because we are using learning rate scheduler which will change the learning rate after every batch of training after each epoch.

13)Defining the final architecture of our model

```
# base class for the model
class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate prediction
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {"val_loss": loss.detach(), "val_accuracy": acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x["val_loss"] for x in outputs]
        batch_accuracy = [x["val_accuracy"] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()       # Combine loss
        epoch_accuracy = torch.stack(batch_accuracy).mean()
        return {"val_loss": epoch_loss, "val_accuracy": epoch_accuracy} # Combine accuracies

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result['val_accuracy']))
```

In [30]:
```
# Architecture for training

# convolution block with BatchNormalization
def ConvBlock(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool:
        layers.append(nn.MaxPool2d(4))
    return nn.Sequential(*layers)


# resnet architecture
class ResNet9(ImageClassificationBase):
    def __init__(self, in_channels, num_diseases):
        super().__init__()

        self.conv1 = ConvBlock(in_channels, 64)
        self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
        self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

        self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16
        self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44
        self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

        self.classifier = nn.Sequential(nn.MaxPool2d(4),
                                        nn.Flatten(),
                                        nn.Linear(512, num_diseases))

    def forward(self, xb): # xb is the loaded batch
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.res1(out) + out
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.res2(out) + out
        out = self.classifier(out)
        return out
```

14) Defining and moving our model into the GPU

In [31]:
```
# defining the model and moving it to the GPU
model = to_device(ResNet9(3, len(train_ds.classes)), device)
model
```

15)Training the model

- **Learning Rate Scheduling**: Instead of using a fixed learning rate, we will use a learning rate scheduler, which will change the learning rate after every batch of training. There are many strategies for varying the learning rate during training, and the one we'll use is called the *"One Cycle Learning Rate Policy"*, which involves starting with a low learning rate, gradually increasing it batch-by-batch to a high learning rate for about 30% of epochs, then gradually decreasing it to a very low value for the remaining epochs.
- **Weight Decay**: We also use weight decay, which is a regularization technique which prevents the weights from becoming too large by adding an additional term to the loss function.
- **Gradient Clipping**: Apart from the layer weights and outputs, it also helpful to limit the values of gradients to a small range to prevent undesirable changes in parameters due to large gradient values. This simple yet effective technique is called gradient clipping.

We'll also record the learning rate used for each batch.

Training

In [33]:
```
# for training
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)


def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']


def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader, weight_decay=0,
                 grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []

    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # scheduler for one cycle learniing rate
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs, steps_per_epoch=len(train_loader))
```

```
    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        lrs = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # recording and updating learning rates
            lrs.append(get_lr(optimizer))
            sched.step()


        # validation
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)

    return history
```

16) Let's check our validation loss and accuracy (Since there are randomly initialized weights, that is why accuracy come to near 0.0331that is 3.3% chance of getting the right answer or you can say model randomly chooses a class.)

In [34]:
```
# checking validation loss and accuracy
%%time
history = [evaluate(model, valid_ddl)]
history
```

```
CPU times: user 45.5 s, sys: 4.57 s, total: 50.1 s
Wall time: 1min 11s
```

Out[34]:
```
[{'val_loss': tensor(3.6390, device='cuda:0'), 'val_accuracy': tensor(0.0331)}]
```

The accuracy was 3.31%, as random weights were assigned.

17) Now, declare some hyper parameters for the training of the model. We can change it if result is not satisfactory

Assigned values

In [35]:
```
epochs = 2
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

18) Let's start training our model ....

Note: The following cell may take 15 mins to 45 mins to run depending on your GPU.

In [36]:
```
%%time
history += fit_OneCycle(epochs, max_lr, model, train_ddl, valid_ddl,
                            grad_clip=grad_clip,
                            weight_decay=1e-4,
                            opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00812, train_loss: 0.7356, val_loss: 0.7064, val_acc: 0.8007
Epoch [1], last_lr: 0.00000, train_loss: 0.1245, val_loss: 0.0256, val_acc: 0.9920
CPU times: user 14min 29s, sys: 9min 26s, total: 23min 55s
Wall time: 24min 54s
```

Accuracy is around 99.2%

19)Plot

```
def plot_accuracies(history):
    accuracies = [x['val_accuracy'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');
```

```
def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');
```

```
def plot_lrs(history):
    lrs = np.concatenate([x.get('lrs', []) for x in history])
    plt.plot(lrs)
    plt.xlabel('Batch no.')
    plt.ylabel('Learning rate')
    plt.title('Learning Rate vs. Batch no.');
```

20)Saving the entire model to working directory

Saving the Model

saving the models 'state_dict' using 'torch.save'

In [51]:
```
# saving to the working directory
loc = '../Models/plant-disease-model.pth'
torch.save(model.state_dict(), loc)
```

ACCURACY

We have got the accuracy of 99.2% after training the model.

In [36]:
```
%%time
history += fit_OneCycle(epochs, max_lr, model, train_ddl, valid_ddl,
                        grad_clip=grad_clip,
                        weight_decay=1e-4,
                        opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00812, train_loss: 0.7356, val_loss: 0.7064, val_acc: 0.8007
Epoch [1], last_lr: 0.00000, train_loss: 0.1245, val_loss: 0.0256, val_acc: 0.9920
CPU times: user 14min 29s, sys: 9min 26s, total: 23min 55s
Wall time: 24min 54s
```

Accuracy is around 99.2%

TESTING

PREDICTIONS

Some of our predictions:

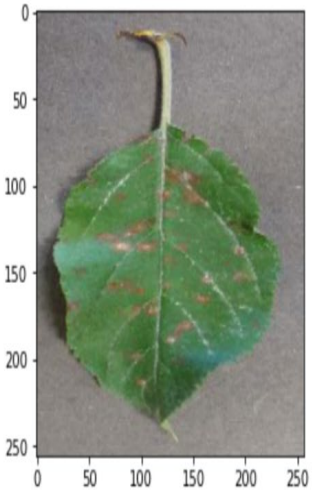
Testing with individual images

There is a test dir with 33 diff images to test

In [43]:
```
test_dir = "../Datasets/Imgs/test/test"
test = ImageFolder(test_dir, transform=transforms.ToTensor())
```

In [48]:
```
img, label = test[1]
plt.imshow(img.permute(1, 2, 0))
print('Label:', test_images[0], ', Predicted:', predict_image(img, model))
```

```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
```

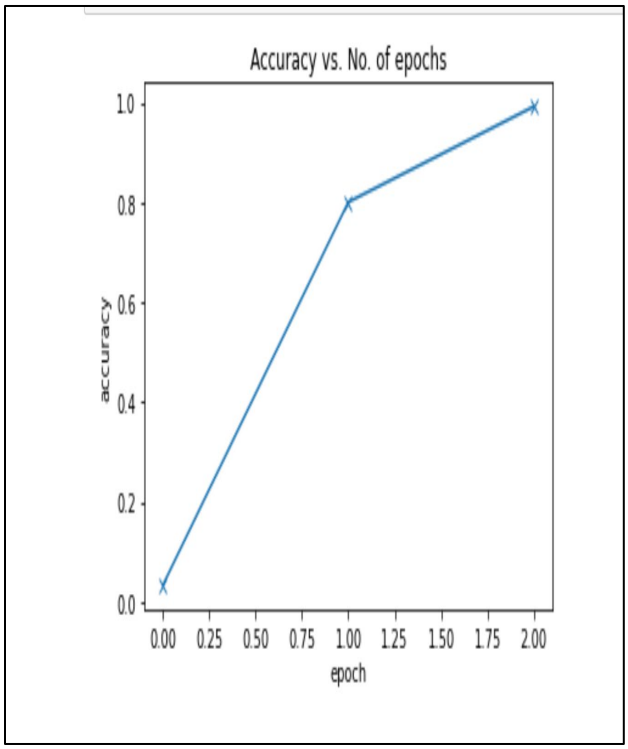


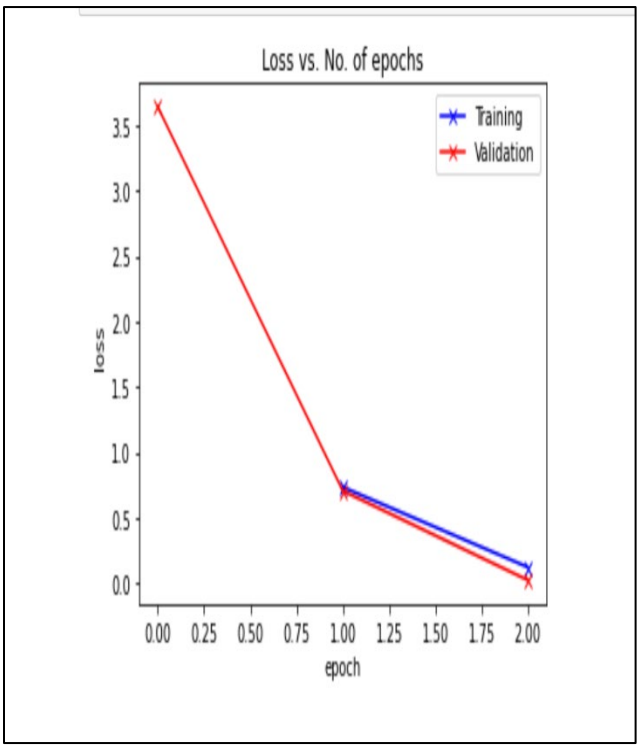
In [47]:
```
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, model))
```

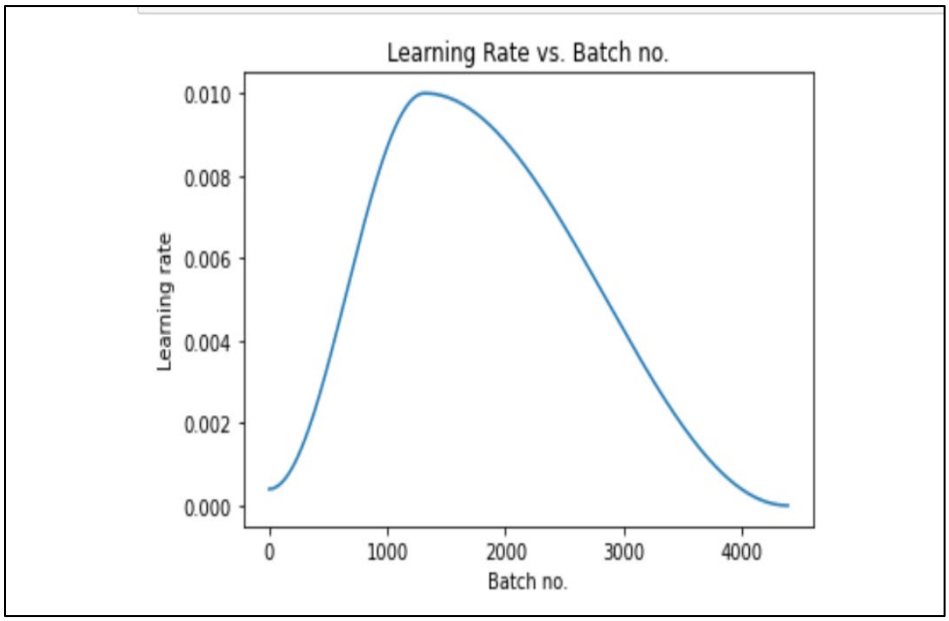
```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust2.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust3.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust4.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleScab1.JPG , Predicted: Apple___Apple_scab
Label: AppleScab2.JPG , Predicted: Apple___Apple_scab
Label: AppleScab3.JPG , Predicted: Apple___Apple_scab
Label: CornCommonRust1.JPG , Predicted: Corn_(maize)___Common_rust_
Label: CornCommonRust2.JPG , Predicted: Corn_(maize)___Common_rust_
Label: CornCommonRust3.JPG , Predicted: Corn_(maize)___Common_rust_
Label: PotatoEarlyBlight1.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight2.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight3.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight4.JPG , Predicted: Potato___Early_blight
Label: PotatoEarlyBlight5.JPG , Predicted: Potato___Early_blight
Label: PotatoHealthy1.JPG , Predicted: Potato___healthy
Label: PotatoHealthy2.JPG , Predicted: Potato___healthy
Label: TomatoEarlyBlight1.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight2.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight3.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight4.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight5.JPG , Predicted: Tomato___Early_blight
Label: TomatoEarlyBlight6.JPG , Predicted: Tomato___Early_blight
Label: TomatoHealthy1.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy2.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy3.JPG , Predicted: Tomato___healthy
Label: TomatoHealthy4.JPG , Predicted: Tomato___healthy
Label: TomatoYellowCurlVirus1.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus2.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus3.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus4.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus5.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Label: TomatoYellowCurlVirus6.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus
```

GRAPHS

Graphs for our data







7.UI (USER INTERFACE)

In-order to simplify the relation between the software and user and for the better communication of information user interface has been developed

FRONTEND

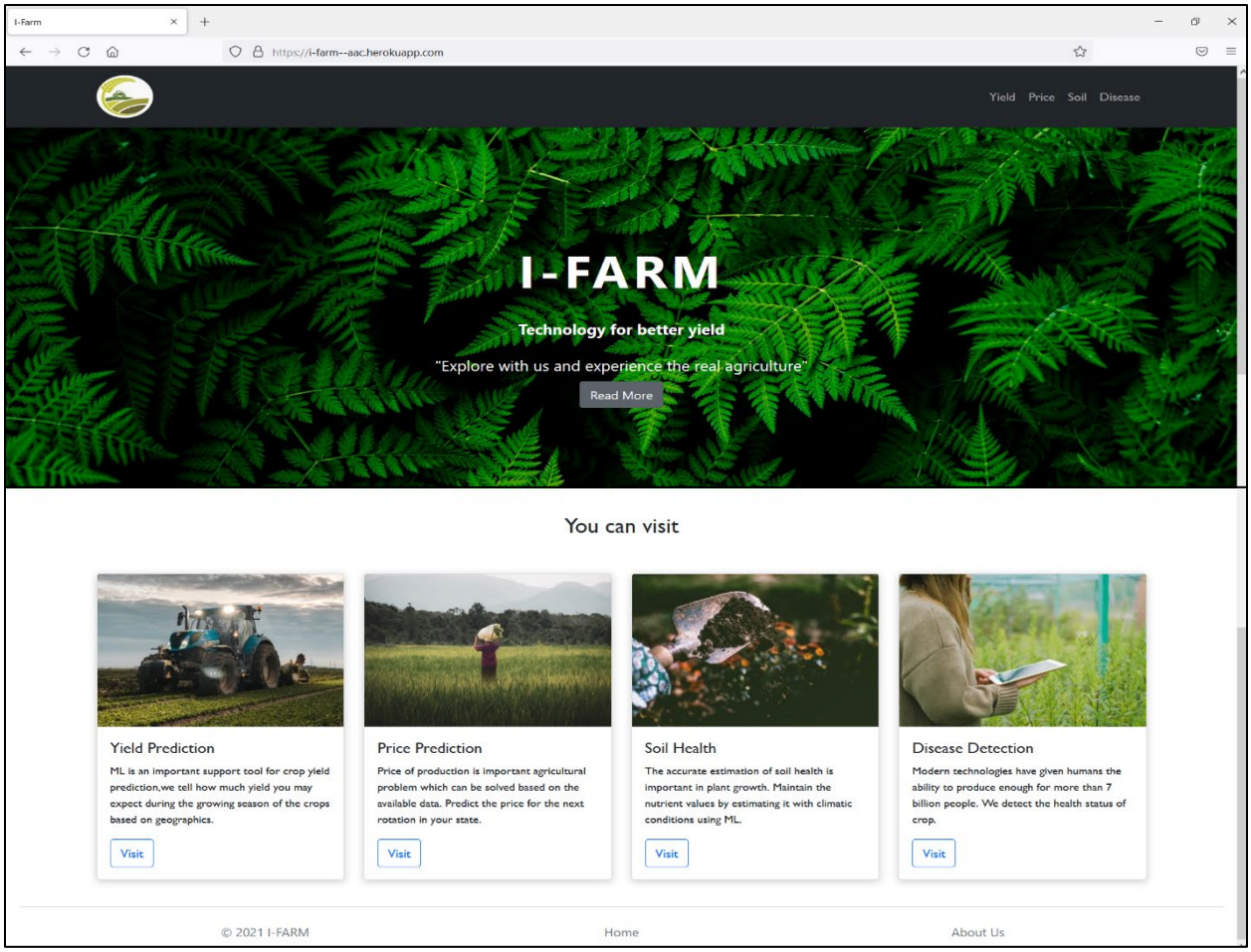
1)HTML

2)CSS

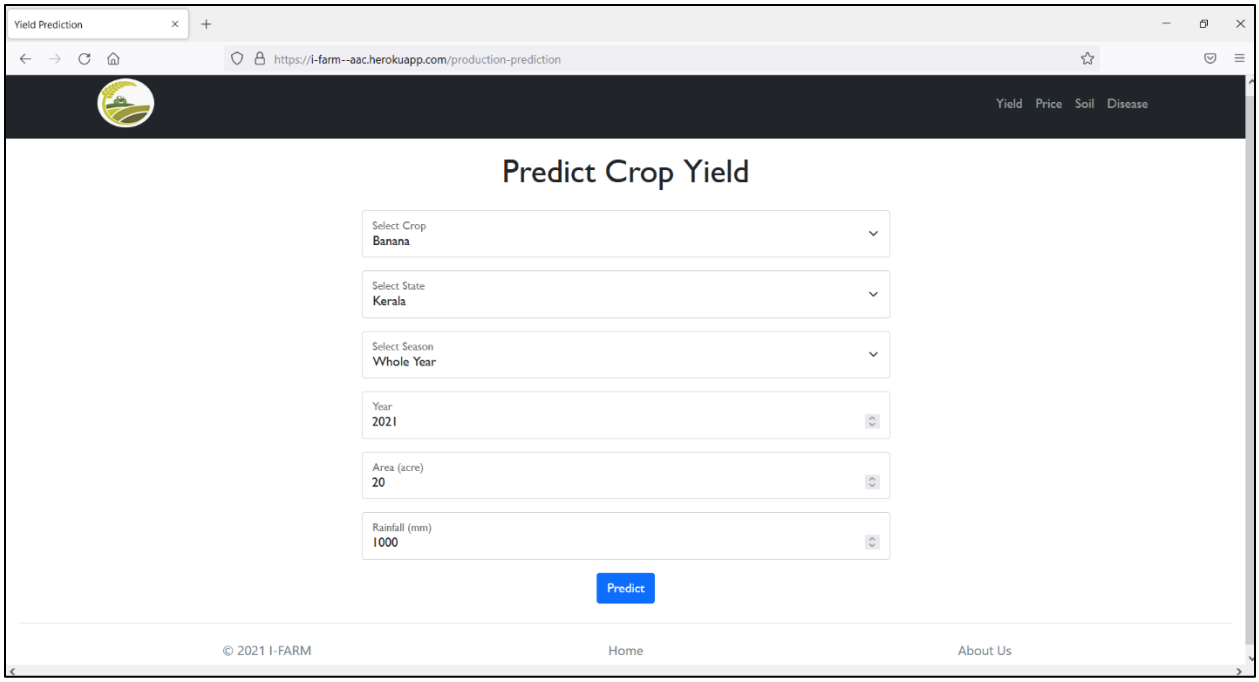
3)Java Script

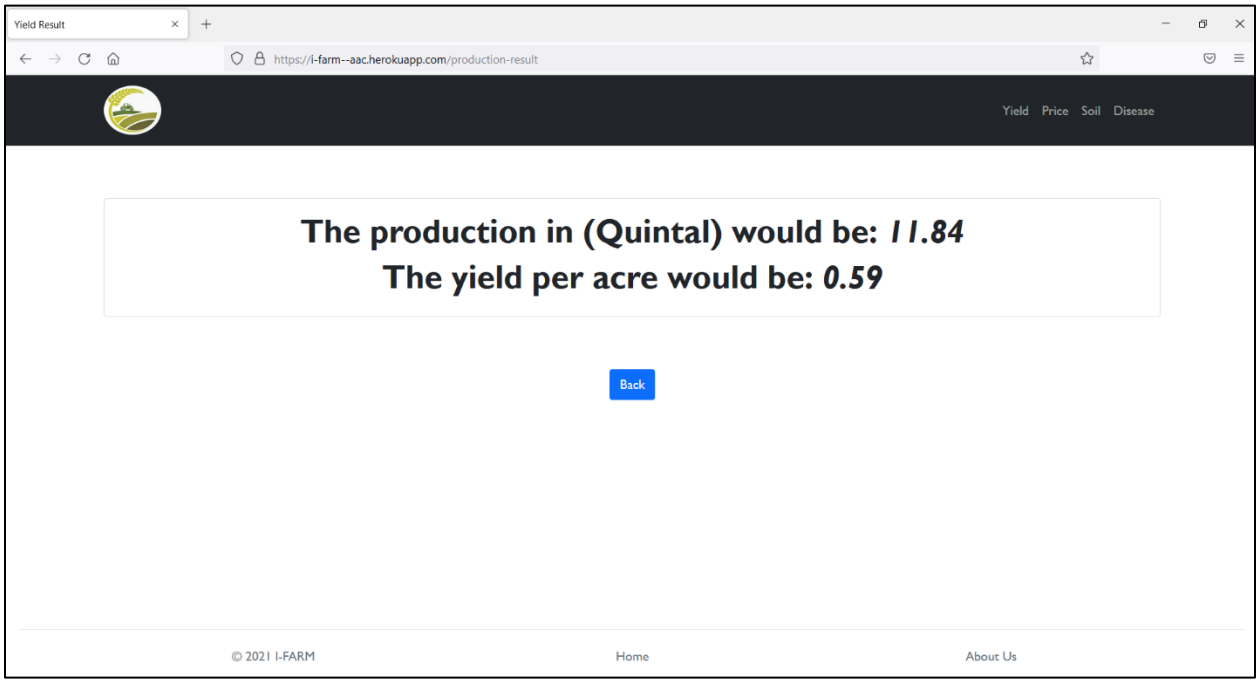
4)Bootstrap

Home page

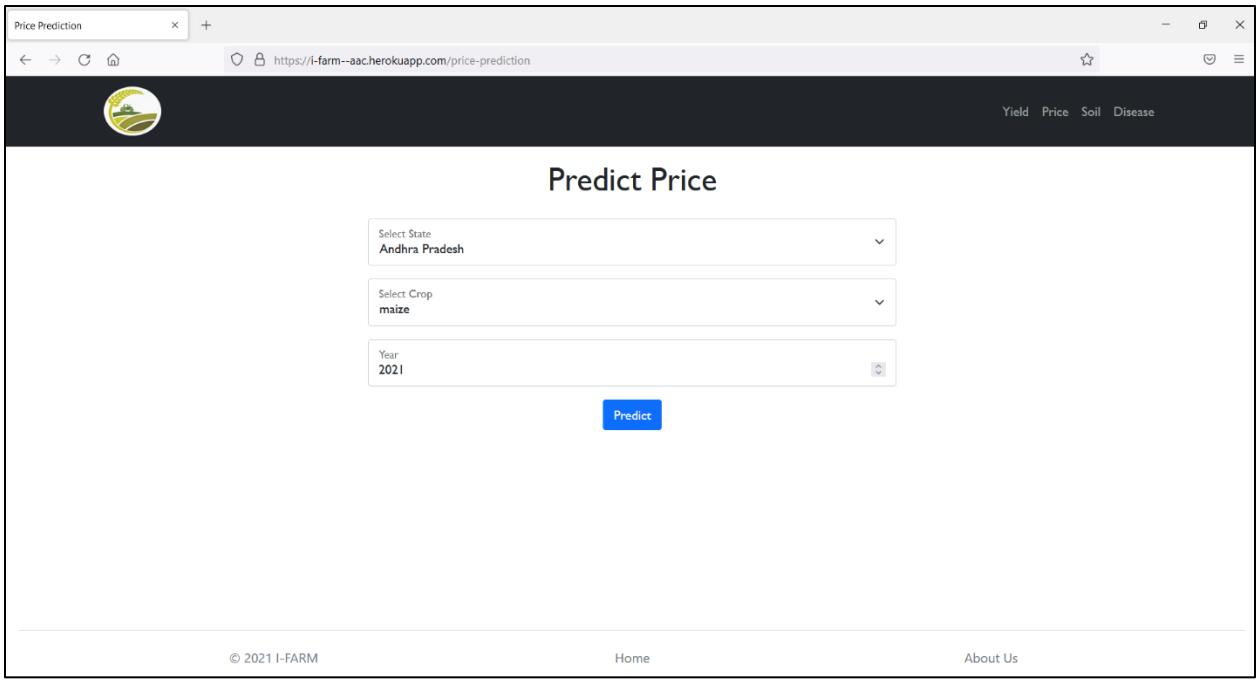


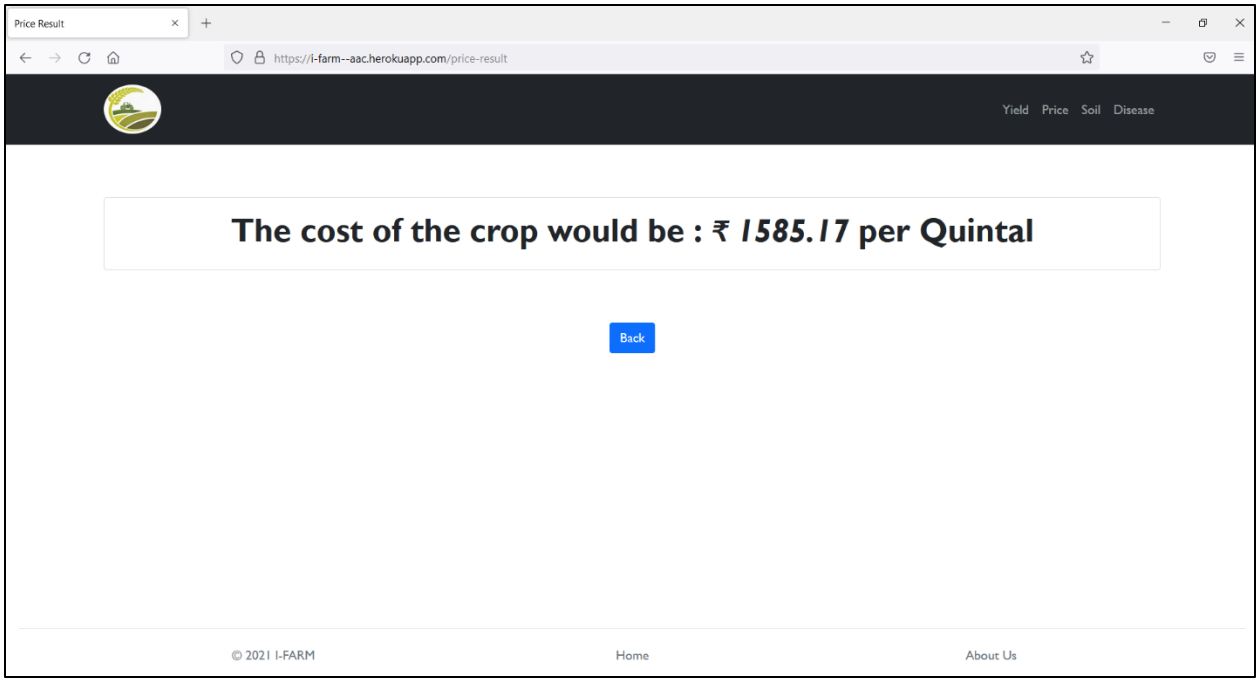
Yield prediction



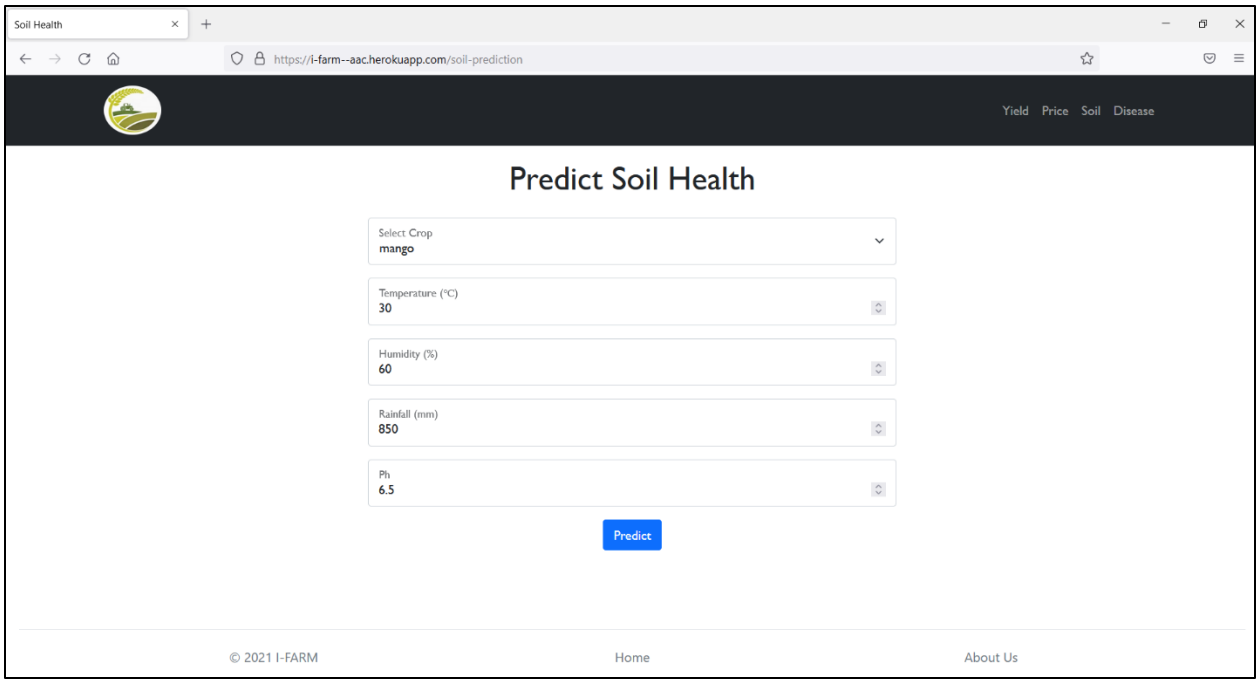


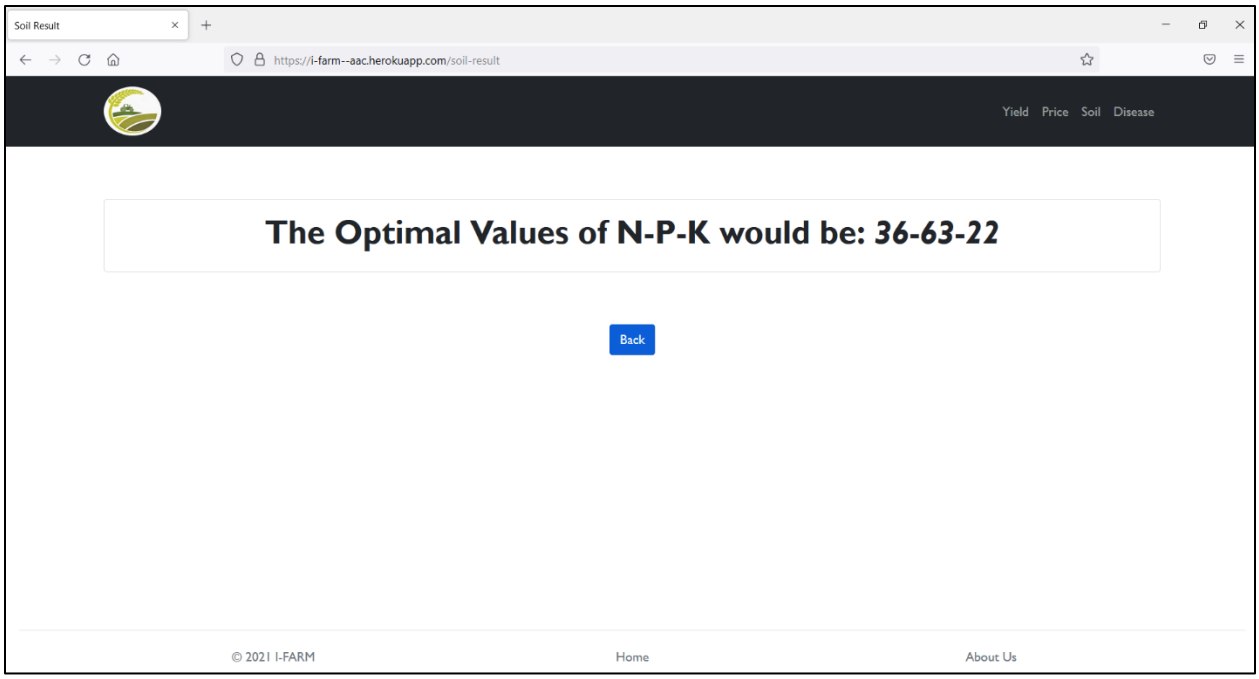
Price Prediction



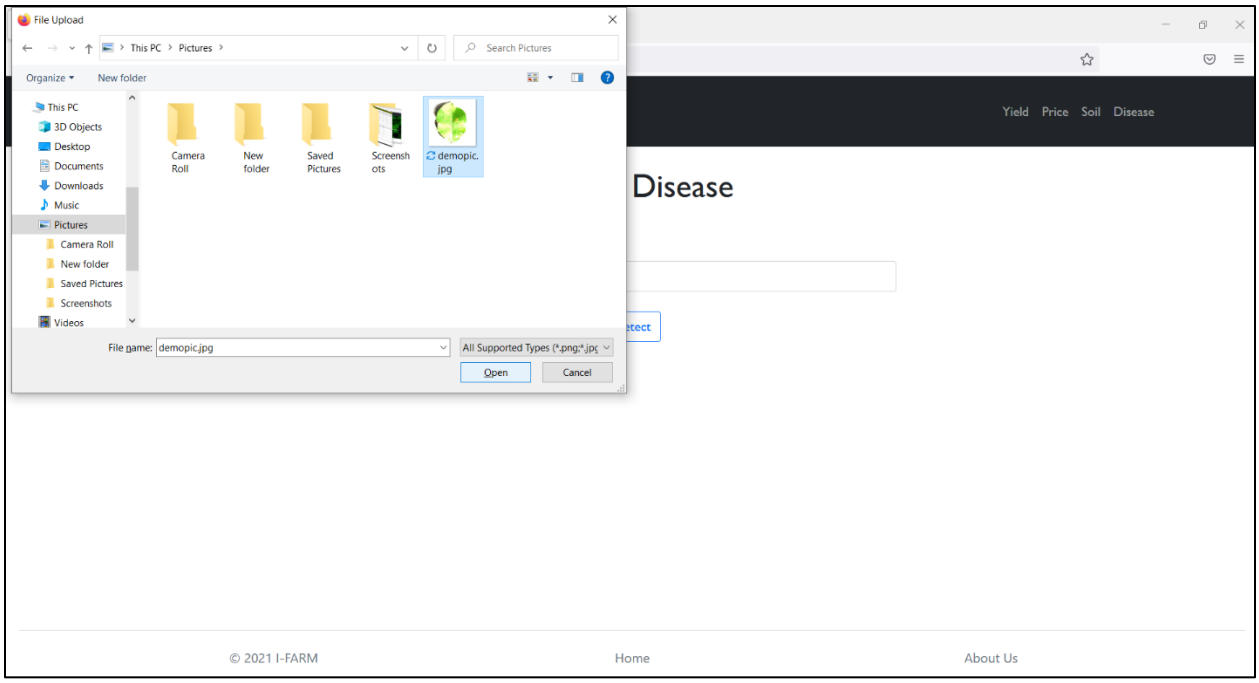


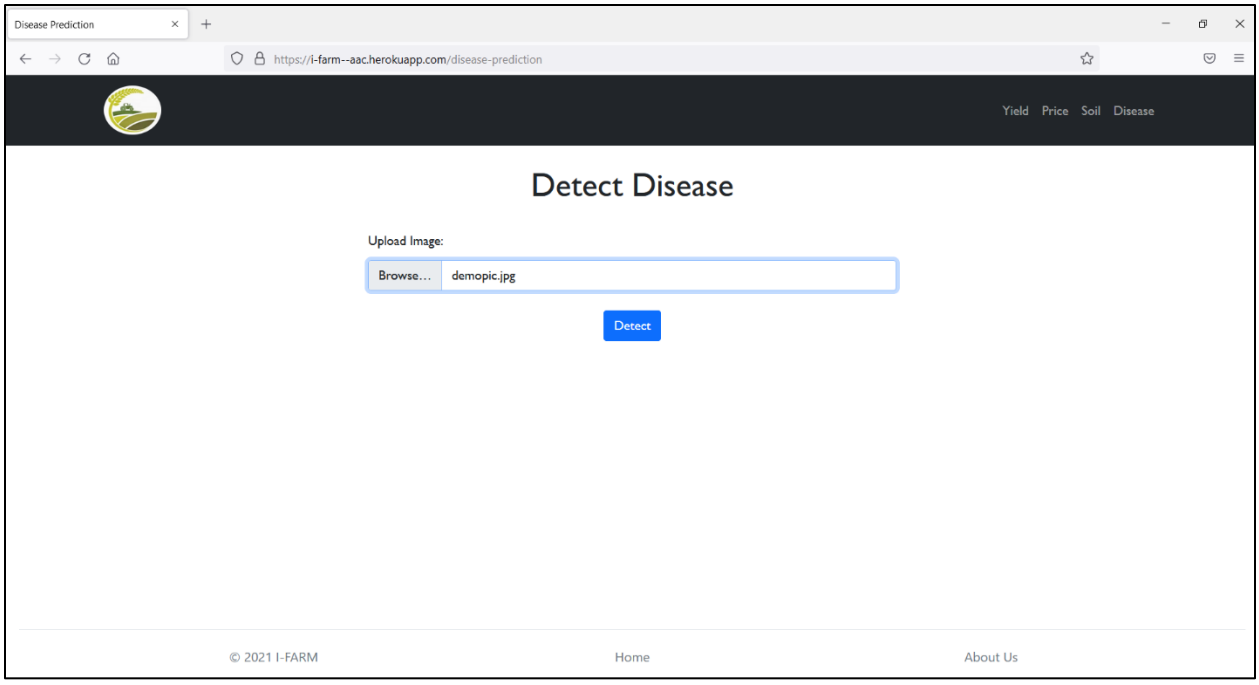
Soil Health Prediction

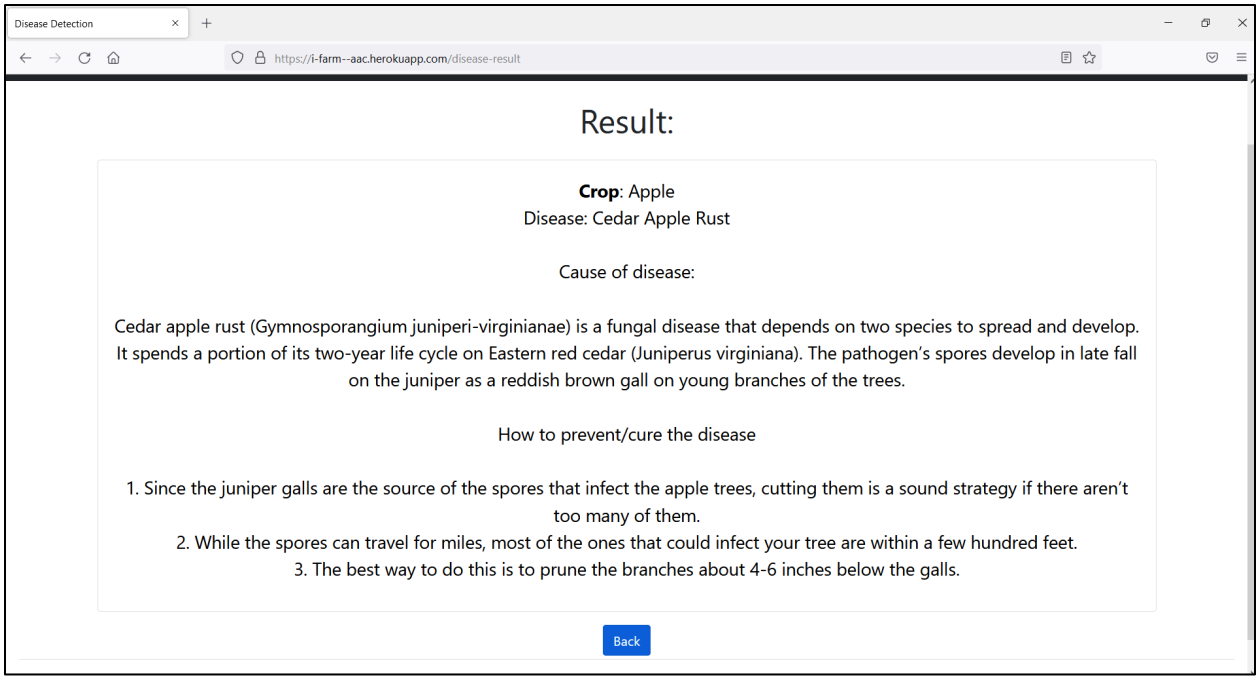




Disease Detection







BACKEND

1)Flask

UI LINK

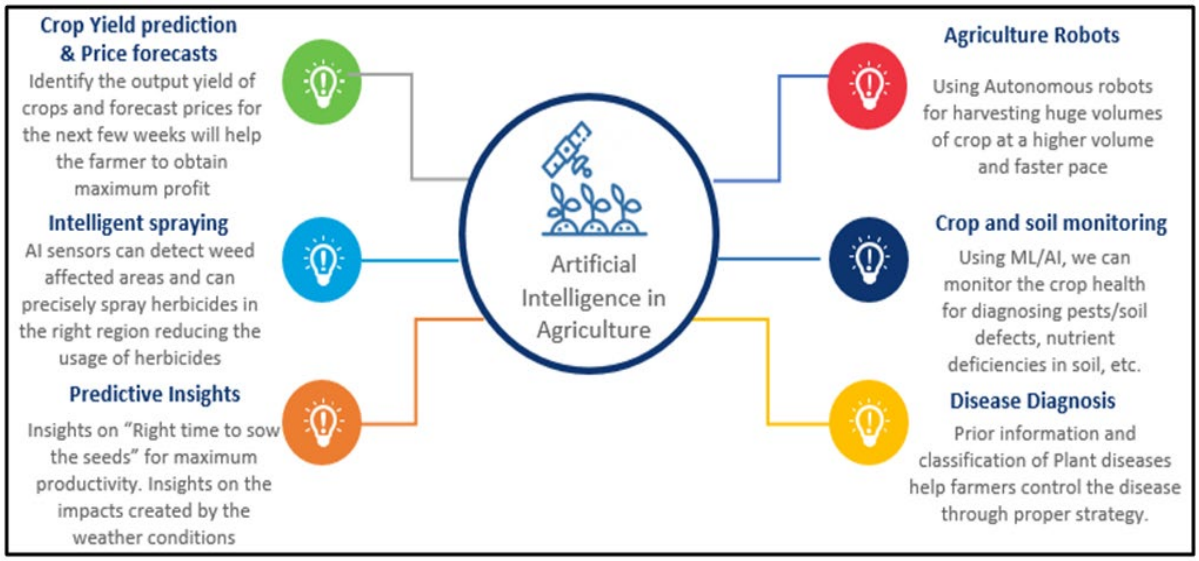
https://i-farm--aac.herokuapp.com/

GITHUB LINK

GitHub link for all our source codes and datasets is given below

https://github.com/Mohit-3430/AAC

9.FUTURE DEVELOPMENTS



1) AI sensors can detect and target weeds and then decide which herbicide to apply within the region. We can use computer vision and artificial intelligence to monitor and precisely spray on weeds.[10]
2) Use of weather forecasting: With help of Artificial Intelligence farmers can analyze weather conditions by using weather forecasting which helps they plan the type of crop can be grown and when should seeds be sown.[10]
3) Analyzing crop health by drones: the drone captures data from fields and then data is transferred via a USB drive from the drone to a computer and analyzed by experts. It helps the farmer to identify pests and bacteria helping farmers to timely use of pest control and other methods to take required action.[10]
4) IoT is about the power of data. Our world is digitally connected and data is a critical asset. We can improvise it by remote monitoring of farm conditions and infrastructure, saving time and labour on routine farm checks. improving producers' decision making through data analytics.[11]
5) We can enhance it by predicting the eligibility/possibility for agriculture loan using machine learning algorithms.

10.REFERENCES

1)https://www.irjet.net/archives/V6/i4/IRJET-V6I4483.pdf

2)https://www.questjournals.org/jses/papers/Vol6-issue-1/B06011420.pdf

3)https://towardsdatascience.com/predicting-crops-yield-machine-learning-nanodegree-capstone-project-e6ec9349f69

4)https://www.ijert.org/research/efficient-crop-yield-prediction-in-india-using-machine-learning-techniques-IJERTCONV8IS10005.pdf

5)https://www.apnikheti.com/en/pn/agriculture/horticulture/spice-and-condiments/coriander

6)https://thenutrientcompany.com/blogs/horticulture/npk-value-of-everything-organic-database

7) https://www.pennington.com/all-products/fertilizer/resources/understanding-fertilizer-labels

8)https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full#h3

9) https://plantvillage.psu.edu/plants

10)https://www.analyticsvidhya.com/blog/2020/11/artificial-intelligence-in-agriculture-using-modern-day-ai-to-solve-traditional-farming-problems/

11) https://agriculture.vic.gov.au/farm-management/digital-agriculture/internet-of-things-in-agriculture

12) https://medium.com/swlh/natural-image-classification-using-resnet9-model-6f9dc924cd6d