Step 1 - Goal Selection

UN SDG's Selected :

1) Goal 2 - Zero Hunger

2) Goal 8 - Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

Objectives

Smart Farming Tips: Use this information to figure out the best crop for farmers to grow. It's like giving them smart tips to get more food from their fields.

Technology to Help Farmers: Use computer programs to make predictions about farming. It's a bit like how weather apps predict if it will rain or be sunny.

Financial Help for Farmers: Look at how much money farmers make and spend. Based on that, we helped them get loans or financial support. This way, they can invest in their farms and improve their income.

Putting Everything Together: Combine farming tips and financial help to make decisions that are good for both the farm and the farmer's wallet.

Helping with Loans: If a farmer is doing well with their crops, we might suggest they get a loan to expand their farm or invest in better equipment.

More Food: By giving farmers better advice, we're helping them grow more food. This is important to make sure everyone has enough to eat.

Less Poverty: By helping farmers with money matters, we're also trying to reduce poverty. If farmers make more money, they can have a better life and help their communities.

Making the World Better: We're trying to make the world a better place by using technology and smart ideas to help farmers and improve how we grow our food. In simple terms, it's like we're using computer smarts to help farmers grow more food, make more money, and make the world a better place for everyone.

Step 2 - Data Integration & Processing

Importing the libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import requests
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

Importing Dataset

```
crop = pd.read_csv('Farmerly.csv')
X = crop.iloc[:, :-1].values
y = crop.iloc[:, -1].values
```

Step 3 - Data Cleaning:

Handling missing data

```
crop = pd.read_csv("Farmerly.csv")
crop.head()
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

```
crop.describe()
```

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

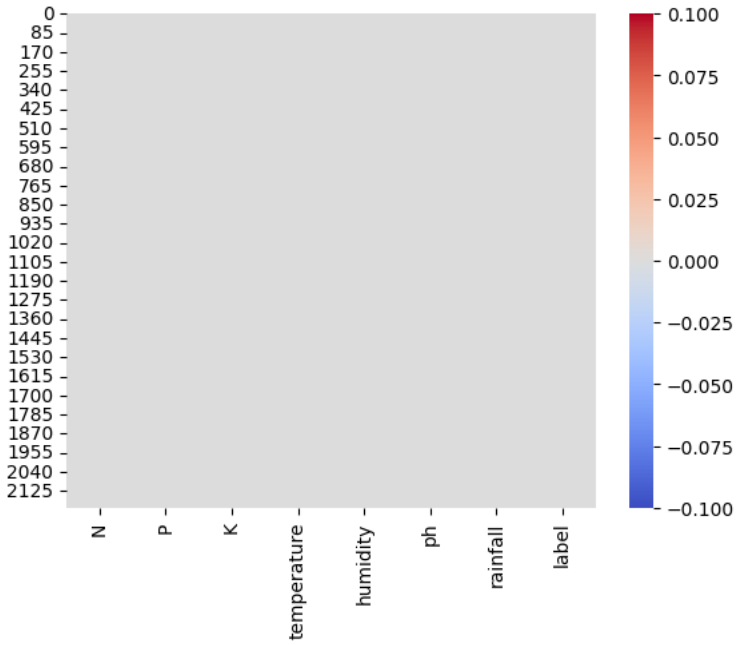
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')

imputer.fit(X[:, 0:1])
X[: , 0:1] = imputer.transform(X[:, 0:1])
```

Exploratory Data Analysis

Heatmap to check null/missing values

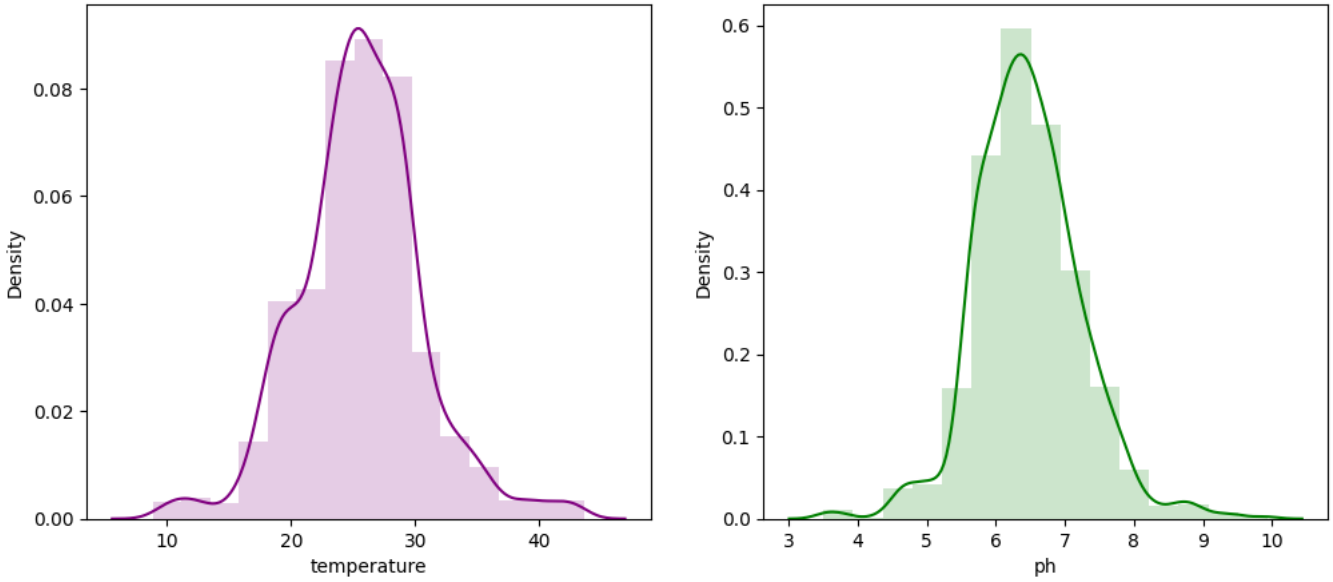
```
sns.heatmap(crop.isnull(),cmap="coolwarm")
plt.show()
```



Distribution of temperature and PH

```
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
sns.distplot(crop['temperature'],color="purple",bins=15,hist_kws={'alpha':0.2})
plt.subplot(1, 2, 2)
sns.distplot(crop['ph'],color="green",bins=15,hist_kws={'alpha':0.2})
```

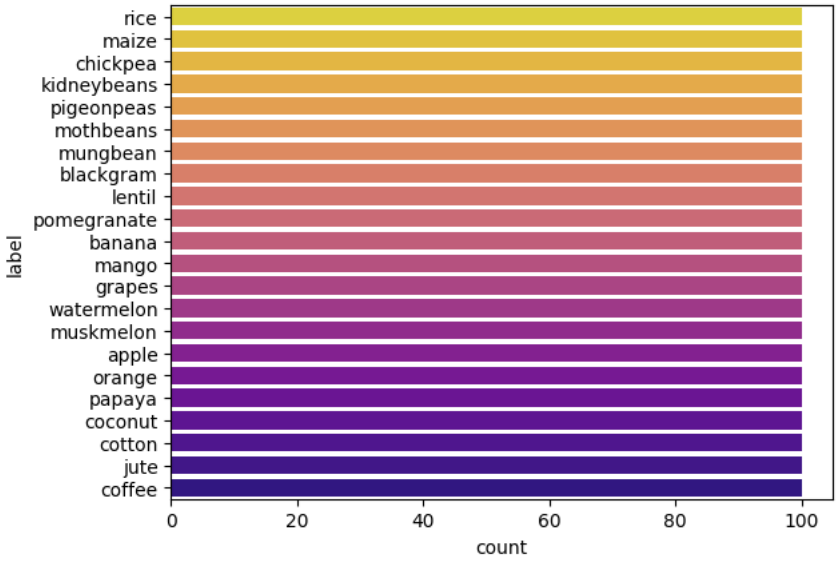
```
<Axes: xlabel='ph', ylabel='Density'>
```



Visualize what Data says

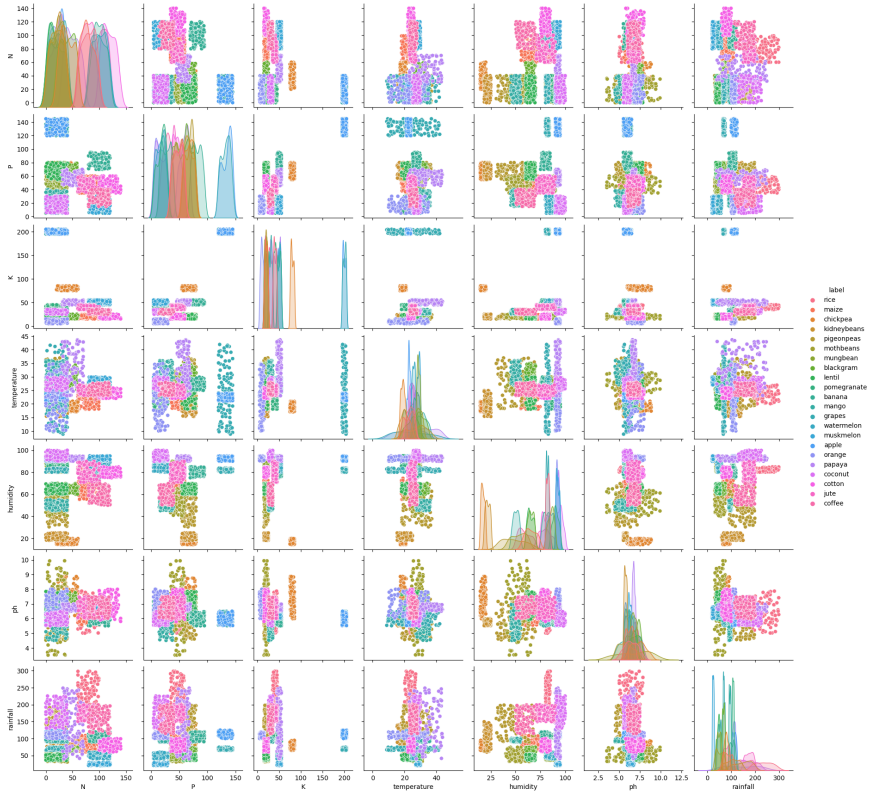
```
sns.countplot(y='label',data=crop, palette="plasma_r")
```

```
<Axes: xlabel='count', ylabel='label'>
```




```
sns.pairplot(crop, hue = 'label')
```

```
<seaborn.axisgrid.PairGrid at 0x797fafa269e0>
```



During rainy season, average rainfall is high (average 120 mm) and temperature is mildly chill (less than 30'C). Rain affects soil moisture which affects ph of the soil. Here are the crops which are likely to be planted during this season. Rice needs heavy rainfall (>200 mm) and a humidity above 80%. No wonder major rice production in India comes from East Coasts which has average of 220 mm rainfall every year! Coconut is a tropical crop and needs high humidity therefore explaining massive exports from coastal areas around the country.

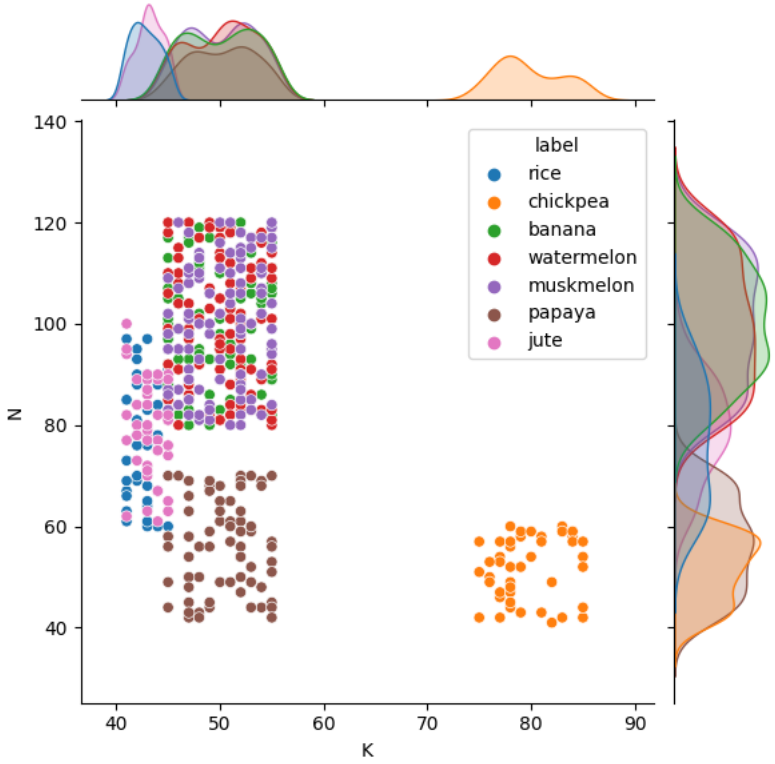
```
sns.jointplot(x="rainfall",y="humidity",data=crop[(crop['temperature']<30) & (crop['rainfall']>120)],hue="label")
```

```
<seaborn.axisgrid.JointGrid at 0x797f7378fac0>
```

This graph correlates with average potassium (K) and average nitrogen (N) value (both>50). These soil ingredients direcly affects nutrition value of the food. Fruits which have high nutrients typically has consistent potassium values.

```
sns.jointplot(x="K",y="N",data=crop[(crop['N']>40)&(crop['K']>40)],hue="label")
```

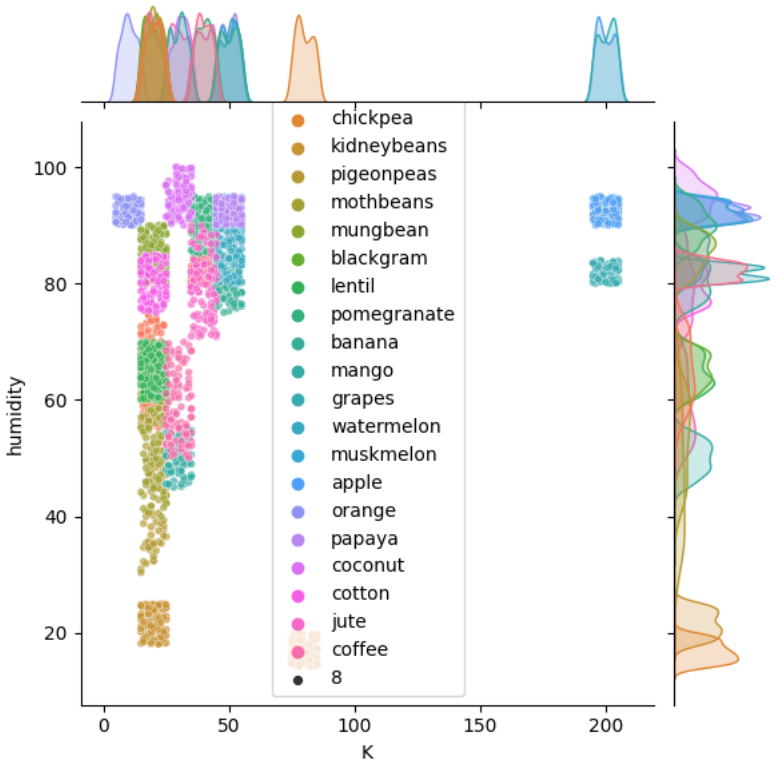
```
<seaborn.axisgrid.JointGrid at 0x797f737a5ea0>
```



sns.jointplot() can be used for bivariate analysis to plot between humidity and K levels based on Label type. It further generates frequency distribution of classes with respect to features

```
sns.jointplot(x="K",y="humidity",data=crop,hue='label',size=8,s=30,alpha=0.7)
```

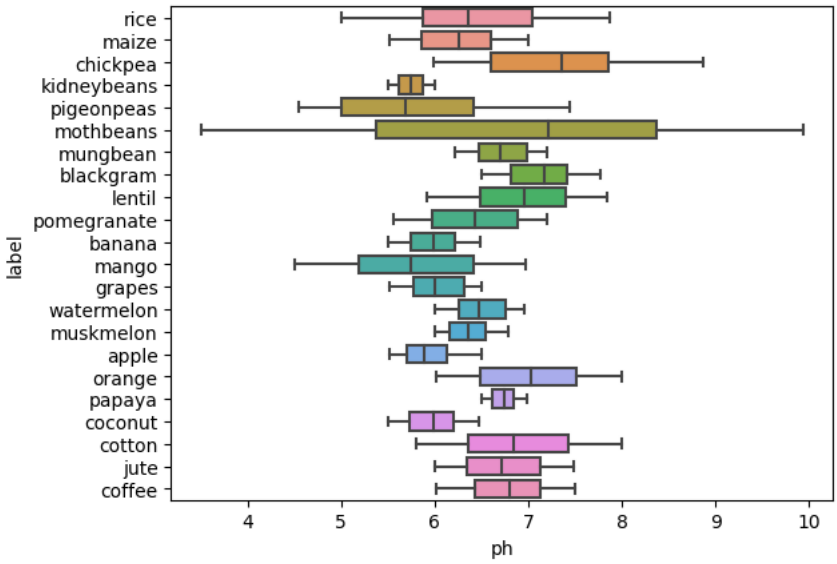
```
<seaborn.axisgrid.JointGrid at 0x797f73add0c0>
```



We can see ph values are critical when it comes to soil. A stability between 6 and 7 is preffered

```
sns.boxplot(y='label',x='ph',data=crop)
```

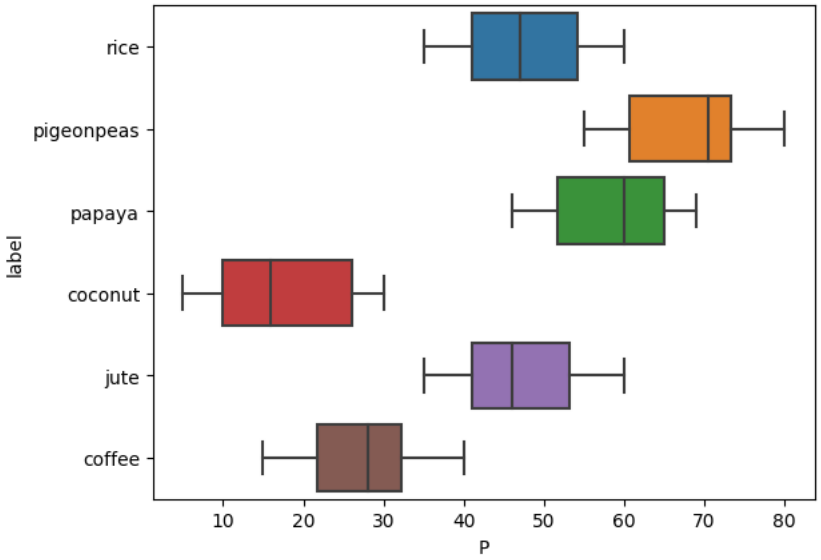
```
<Axes: xlabel='ph', ylabel='label'>
```



Phosphorous levels are quite differentiable when it rains heavily (above 150 mm)

```
sns.boxplot(y='label',x='P',data=crop[crop['rainfall']>150])
```

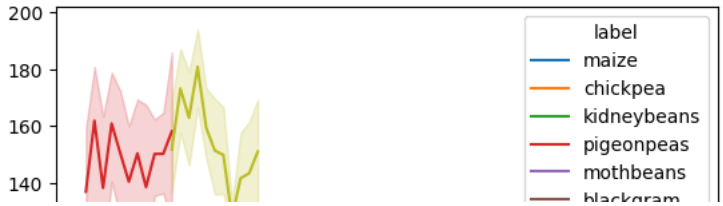
```
<Axes: xlabel='P', ylabel='label'>
```



When humidity is less than 65, almost same phosphor levels(approx 14 to 25) are required for 6 crops which could be grown just based on the amount of rain expected over the next few weeks

```
sns.lineplot(data = crop[(crop['humidity']<65)], x = "K", y = "rainfall",hue="label")
```

```
<Axes: xlabel='K', ylabel='rainfall'>
```



Step 4 - Data Encoding

```
crop_dict = {
    'rice': 1,
    'maize': 2,
    'jute': 3,
    'cotton': 4,
    'coconut': 5,
    'papaya': 6,
    'orange': 7,
    'apple': 8,
    'muskmelon': 9,
    'watermelon': 10,
    'grapes': 11,
    'mango': 12,
    'banana': 13,
    'pomegranate': 14,
    'lentil': 15,
    'blackgram': 16,
    'mungbean': 17,
    'mothbeans': 18,
    'pigeonpeas': 19,
    'kidneybeans': 20,
    'chickpea': 21,
    'coffee': 22
}
crop['crop_num']=crop['label'].map(crop_dict)
```

```
crop['crop_num'].value_counts()
crop.head()
```

	N	P	K	temperature	humidity	ph	rainfall	label	crop_num
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice	1
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice	1
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice	1
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice	1
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice	1

Step 5 - Data : Crop, Split

Cropping Data

```
X = crop.drop(['crop_num','label'],axis=1)
y = crop['crop_num']
```

```
print(X)
```

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

```
[2200 rows x 7 columns]
```

Split - Training set, Testing set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape
```

```
(1760, 7)
```

```
X_test.shape
```

```
(440, 7)
```

```
X_train
```

	N	P	K	temperature	humidity	ph	rainfall
1656	17	16	14	16.396243	92.181519	6.625539	102.944161
752	37	79	19	27.543848	69.347863	7.143943	69.408782
892	7	73	25	27.521856	63.132153	7.288057	45.208411
1041	101	70	48	25.360592	75.031933	6.012697	116.553145
1179	0	17	30	35.474783	47.972305	6.279134	97.790725
...	...	...	...	...	...	...	...
1638	10	5	5	21.213070	91.353492	7.817846	112.983436
1095	108	94	47	27.359116	84.546250	6.387431	90.812505
1130	11	36	31	27.920633	51.779659	6.475449	100.258567
1294	11	124	204	13.429886	80.066340	6.361141	71.400430
860	32	78	22	23.970814	62.355576	7.007038	53.409060

1760 rows × 7 columns

Step 6 - Feature Scaling

Scaling

Min Max Scaler

```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()

X_train = ms.fit_transform(X_train)
X_test = ms.transform(X_test)
```

```
X_train
```

```
array([[0.12142857, 0.07857143, 0.045     , ..., 0.9089898 , 0.48532225,
        0.29685161],
       [0.26428571, 0.52857143, 0.07      , ..., 0.64257946, 0.56594073,
        0.17630752],
       [0.05      , 0.48571429, 0.1       , ..., 0.57005802, 0.58835229,
        0.08931844],
       ...,
       [0.07857143, 0.22142857, 0.13      , ..., 0.43760347, 0.46198144,
        0.28719815],
       [0.07857143, 0.85      , 0.995     , ..., 0.76763665, 0.44420505,
        0.18346657],
       [0.22857143, 0.52142857, 0.085     , ..., 0.56099735, 0.54465022,
        0.11879596]])
```

Standardization

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()


sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[-9.03426596e-01, -1.12616170e+00, -6.68506601e-01, ...,
         9.36586183e-01,  1.93473784e-01,  5.14970176e-03],
       [-3.67051340e-01,  7.70358846e-01, -5.70589522e-01, ...,
        -1.00470485e-01,  8.63917548e-01, -6.05290566e-01],
       [-1.17161422e+00,  5.89737842e-01, -4.53089028e-01, ...,
        -3.82774991e-01,  1.05029771e+00, -1.04580687e+00],
       ...,
       [-1.06433917e+00, -5.24091685e-01, -3.35588533e-01, ...,
        -8.98381379e-01, -6.34357580e-04, -4.37358211e-02],
       [-1.06433917e+00,  2.12501638e+00,  3.05234239e+00, ...,
         3.86340190e-01, -1.48467347e-01, -5.69036842e-01],
       [-5.01145154e-01,  7.40255346e-01, -5.11839275e-01, ...,
        -4.18045489e-01,  6.86860180e-01, -8.96531475e-01]])
```

Step 7 - Machine Learning Model

Evaluating the different Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machine': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Bagging': BaggingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Extra Trees': ExtraTreeClassifier(),
}


for name, md in models.items():
    md.fit(X_train,y_train)
    ypred = md.predict(X_test)

    print(f"{name}  with accuracy : {accuracy_score(y_test,ypred)}")
```

```
Logistic Regression  with accuracy : 0.9636363636363636
Support Vector Machine  with accuracy : 0.9681818181818181
K-Nearest Neighbors  with accuracy : 0.9590909090909091
Decision Tree  with accuracy : 0.9840909090909091
Random Forest  with accuracy : 0.9931818181818182
Bagging  with accuracy : 0.990909090909091
AdaBoost  with accuracy : 0.1409090909090909
Gradient Boosting  with accuracy : 0.9818181818181818
Extra Trees  with accuracy : 0.9068181818181819
```

Choosing the best Suitable Model

Random Forest

```
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
ypred = rfc.predict(X_test)
accuracy_score(y_test,ypred)
```

```
0.9931818181818182
```

Step 8 - Predictive Analysis

```
def recommendation(N,P,k,temperature,humidity,ph,rainfal):
    features = np.array([[N,P,K,temperature,humidity,ph,rainfal]])
    transformed_features = ms.fit_transform(features)
    transformed_features = sc.fit_transform(transformed_features)
    prediction = rfc.predict(transformed_features).reshape(1,-1)

    return prediction[0]
```

```
from os import kill
N = float(input("Enter N: "))
P = float(input("Enter P: "))
K = float(input("Enter K: "))
temperature = float(input("Enter temperature: "))
humidity = float(input("Enter humidity: "))
ph = float(input("Enter pH: "))
rainfall = float(input("Enter rainfall: "))

predict = recommendation(N, P, K, temperature, humidity, ph, rainfall)

if predict[0] in crop_dict:
    crop = crop_dict[predict[0]]
    print("{} is the recommended crop for cultivation.".format(crop))
else:
    print("Sorry, we are not able to recommend a proper crop for this environment.")
```

```
Enter N: 83
Enter P: 45
Enter K: 21
Enter temperature: 18.83344
Enter humidity: 58.75082
Enter pH: 5.716223
Enter rainfall: 79.75329
Sorry, we are not able to recommend a proper crop for this environment.
```

Step 9 - Calculating Yield

```
farmers_data = pd.read_csv('farmers_data.csv')
merged_data = pd.merge(crop, farmers_data, on='crop', how='inner')
```

```
X_yield = merged_data.drop(['crop_num', 'label', 'yield'], axis=1)
y_yield = merged_data['yield']

X_yield_train, X_yield_test, y_yield_train, y_yield_test = train_test_split(X_yield, y_yield, test_size=0.2, random_state=42)

rfc_yield = RandomForestClassifier()
rfc_yield.fit(X_yield_train, y_yield_train)

yield_predictions = rfc_yield.predict(X_yield_test)
accuracy_yield = accuracy_score(y_yield_test, yield_predictions)
print(f"Accuracy for yield prediction: {accuracy_yield}")
```

Step 10 - Calculating Credit

```
farmers_data['credit'] = farmers_data['yield'] * farmers_data['total_area']
farmers_data['scaled_credit'] = (farmers_data['credit'] - farmers_data['credit'].min()) / (farmers_data['credit'].max() - farme
```

Future Scope - Website Integration

Dumping Results

```
import pickle
pickle.dump(rfc,open('model.pkl','wb'))
pickle.dump(ms,open('minmaxscaler.pkl','wb'))
pickle.dump(sc,open('standscaler.pkl','wb'))
```

Integrate Website & Database

Importing Libraries

```
!pip install flask
!pip install pyngrok

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip ngrok-stable-linux-amd64.zip
```

Create a Flask Application

```
import getpass
authtoken = getpass.getpass("Enter your Ngrok authtoken: ")
!./ngrok authtoken $authtoken
```

```
from flask_ngrok import run_with_ngrok
from flask import Flask, render_template, request
import numpy as np
import pickle

app = Flask(__name__)
run_with_ngrok(app)

model = pickle.load(open('model.pkl', 'rb'))
sc = pickle.load(open('standscaler.pkl', 'rb'))
ms = pickle.load(open('minmaxscaler.pkl', 'rb'))

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

@app.route("/predict", methods=['POST'])
def predict():
    try:
        return render_template('index.html', result=result)
    except Exception as e:
        return render_template('index.html', result="Error processing the request. Please try again.")

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

Integrate Database

```
pip install Flask-SQLAlchemy
pip install mysql-connector-python
```

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://username:password@hostname/database'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class CropData(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    N = db.Column(db.Float)
    P = db.Column(db.Float)
    K = db.Column(db.Float)
    temperature = db.Column(db.Float)
    humidity = db.Column(db.Float)
    ph = db.Column(db.Float)
    rainfall = db.Column(db.Float)
    label = db.Column(db.String(50))
    crop_num = db.Column(db.Integer)

def recommendation(N, P, K, temperature, humidity, ph, rainfall):
    features = CropData(N=N, P=P, K=K, temperature=temperature, humidity=humidity, ph=ph, rainfall=rainfall)
    db.session.add(features)
    db.session.commit()

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

@app.route('/predict', methods=['POST'])
def predict():
    input_data = request.form['input_data']
    return f'Prediction: {prediction}'

if __name__ == '__main__':
    db.create_all()
    app.run(debug=True)
```

```
data_set = CropData.query.all()
```

```
pickle.dump(rfc, open('model.pkl', 'wb'))
pickle.dump(ms, open('minmaxscaler.pkl', 'wb'))
pickle.dump(sc, open('standscaler.pkl', 'wb'))

db.create_all()
```

Containerize the application using Docker

```
FROM python:3.8-slim
WORKDIR /app
COPY . /app
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 80
ENV NAME World
CMD ["python", "app.py"]
```

```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
mysql-connector-python==8.0.27
pyngrok==5.0.6
scikit-learn==0.24.2
```

```
docker build -t Farmerly .
docker run -p 4000:80 Farmerly
```

```
version: '3'
services:
  web:
    build: .
    ports:
      - "4000:80"
```

```
docker-compose up
```

Host the application on Cloud based hosting site (Heroku/AWS)

```
heroku container:login
docker tag Farmerly:latest registry.heroku.com/Farmerly/web
docker push registry.heroku.com/Farmerly/web
heroku container:release web -a Farmerly
```