AGRI – CROP PREDICTOR

DS5006 Machine Learning

Course Project

Team Details

1) Rahul Kumar (212305014@smail.iitpkd.ac.in)

2) Abigairl Nyasha Chigwededza (142402015@smail.iitpkd.ac.in)

3) Y Vishnu Vardhan

(142402016@smail.iitpkd.ac.in)

TABLE OF CONTENTS

INTRODUCTION

In the current situation, world is experiencing a vibrant shift of monsoons and climatic conditions due to global warming. It’s time to rely on predictors especially in the field of Agriculture as it is one of the essential for living being. To tackle the irregularities in the climate as well as soil parameters, we need to adapt efficient machine learning algorithms and plant the appropriate crop in the appropriate lands to get the maximum yield.

We need to initially gather plenty of information in order to feed the model with all ranges of values. As climatic conditions and underneath soil substances vary place to place, we get region specific crop prediction. During season shifts, Agri-Crop Predictors help in changing the type of crop in the fields by calculating the parameters. We can compare Agriculture Statistics over the world and begin effective imports and exports of the crops yield.

Coming to the usage of fertilizers like pesticides so on, we can rely on the machine learning models to get the optimal measurements. So that, we will experience minimal crop wastages. Hence, our team decided to build the machine learning models by fitting dataset containing vital insights like soil parameters, climatic conditions, pesticides information, crop productivity parameters in different countries so on. We also performed Parameter Analysis and derived multiple data insights and inferences.

PROBLEM DEFINITION

Given a dataset (Crop_recommendation.csv) which would allow the users to build a predictive model to recommend the most suitable crops to grow in a particular farm based on various parameters. This dataset was built by augmenting datasets of rainfall, climate and fertilizer data available for India.

Consider another dataset (Crop Yield Prediction Dataset) which was taken from FAO (Food and Agriculture Organization) http://www.fao.org/home/en/ and World Data Bank https://data.worldbank.org/. Data related to Pesticides and Yield are collected from FAO. Rainfall and Avg. Temperature Data are collected from World Data Bank. At the end, Yield_df.csv is the coagulated final dataset processed by cleaning & merging of pesticides, yield, rainfall & average temperature parameters.

We need to perform suitable data preprocessing techniques to grab the data insights and inferences. Concise data analysis to be done on the features of the datasets via finding correlations and interdependencies. Plot the insights with appropriate charts. Examine the learning algorithms performances to the given input. Suggest what changes to be done to the hyperparameters so that the accuracy gets enhanced. Illustrate real time predictions by the built model.

METHODOLOGY

Primary Dataset is a balanced set covering 22 varieties of crops, each getting 100 combinations of soil and climatic parameters. The 8 data fields present are:

N - ratio of Nitrogen content in soil

P - ratio of Phosphorous content in soil

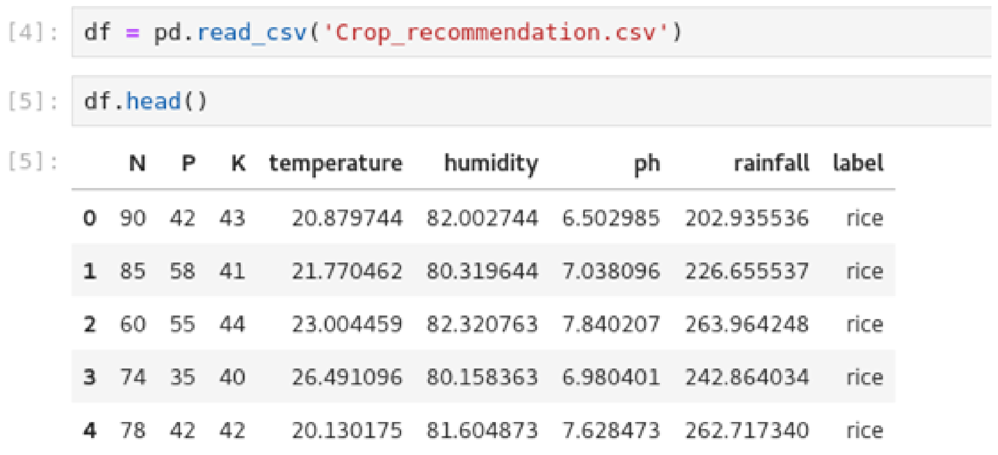
K - ratio of Potassium content in soil

temperature - temperature in degree Celsius

humidity - relative humidity in %

ph - ph value of the soil

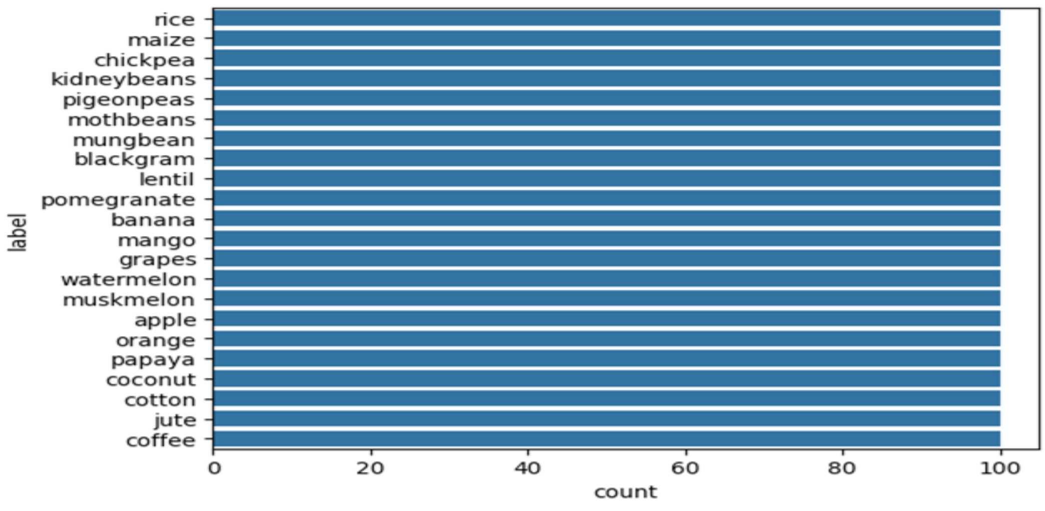
rainfall - rainfall in mm



```
[4]: df = pd.read_csv('Crop_recommendation.csv')

[5]: df.head()

[5]:
```

	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

```
[8]: sns.countplot(y='label',data=df)

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



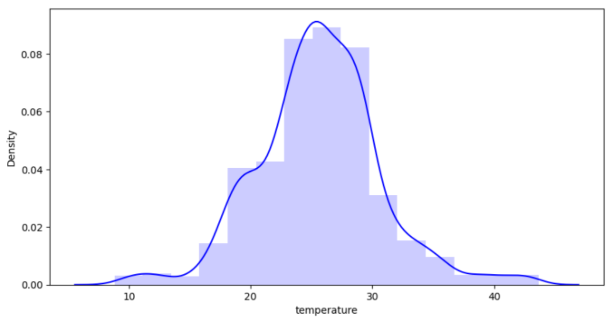
EXPLORATORY DATA ANALYSIS (EDA)

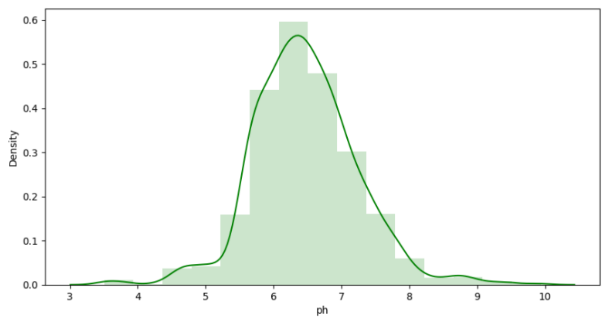
AND INSIGHTS VISUALIZATION

Crop Prediction using soil and weather parameters

Let us view the distribution of Climatic Parameters in the Dataset

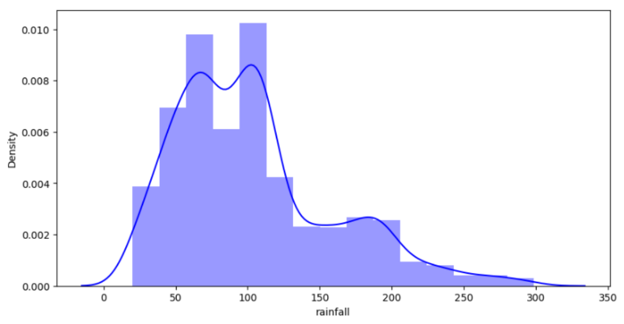
```
[8]:  <Axes: xlabel='ph', ylabel='Density'>
```

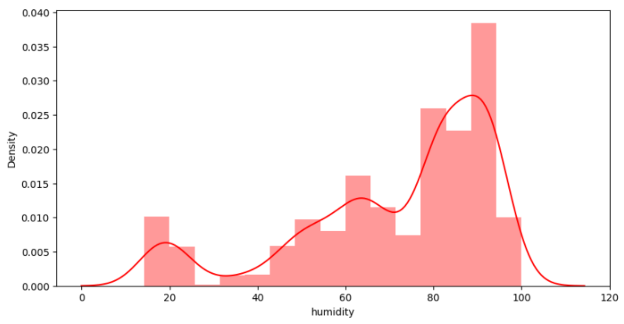




We can infer that both temperature and ph values resembles each other having symmetric, bell-shaped curve concentrated more at the mean centre.

```
[9]:  <Axes: xlabel='humidity', ylabel='Density'>
```

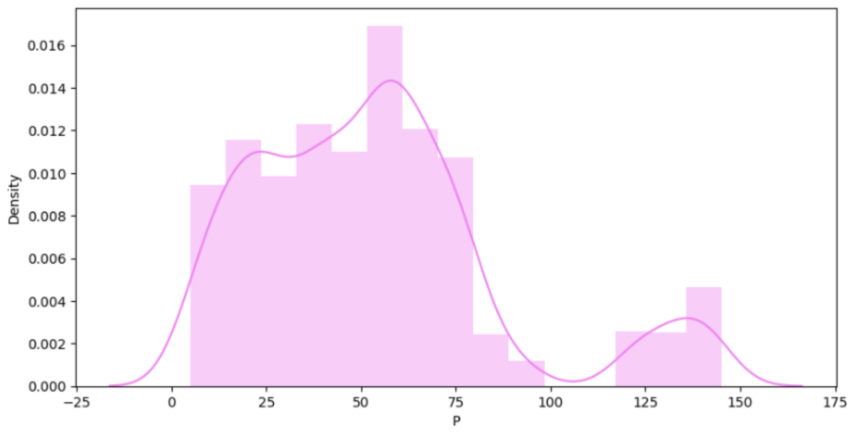


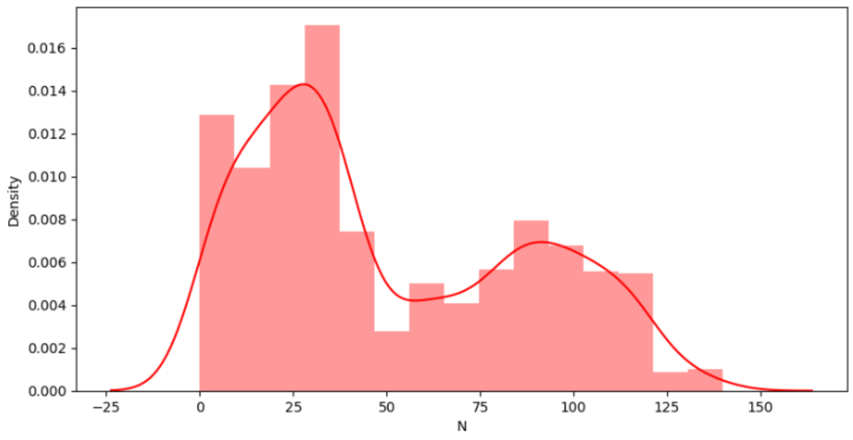


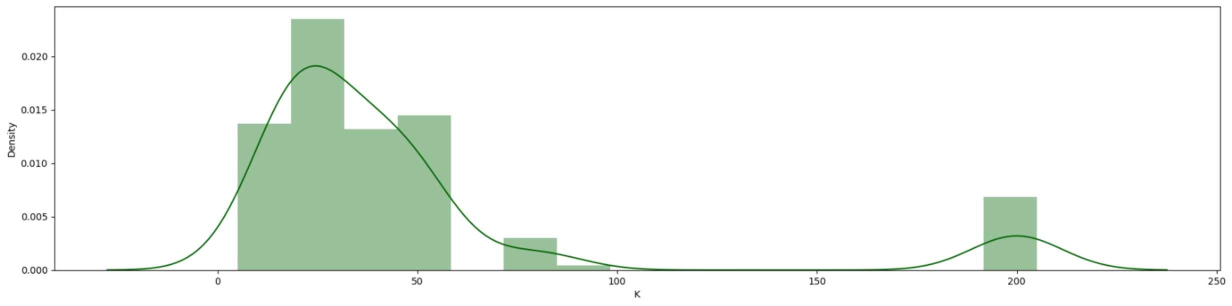
To contrast, rainfall and humidity are not correlated and exhibit different impact

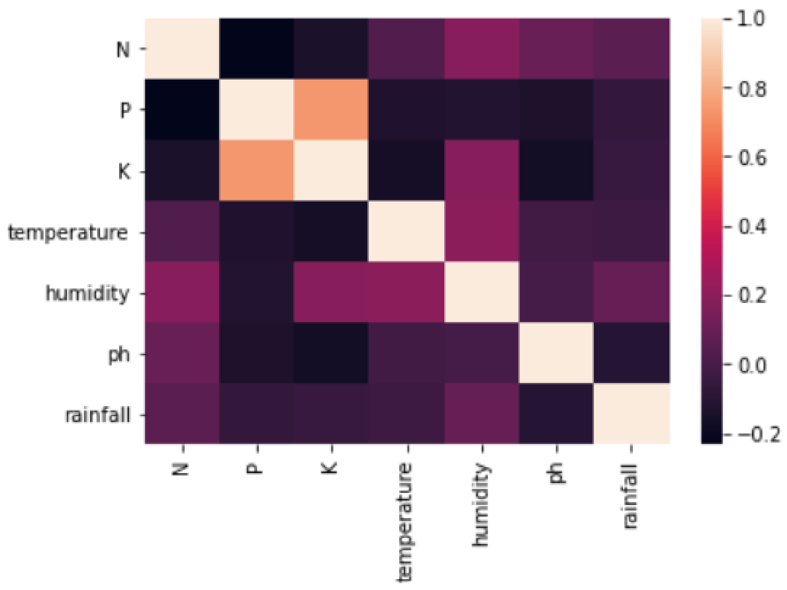
Later stage, we will examine the NPK ratio for given crop and represent them with the help of pie charts.

Now let us view the individual distributions of Nitrogen (N), Phosphorous (P), Potassium (K)


Density
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−25
0
25
50
75
100
125
150
175
P

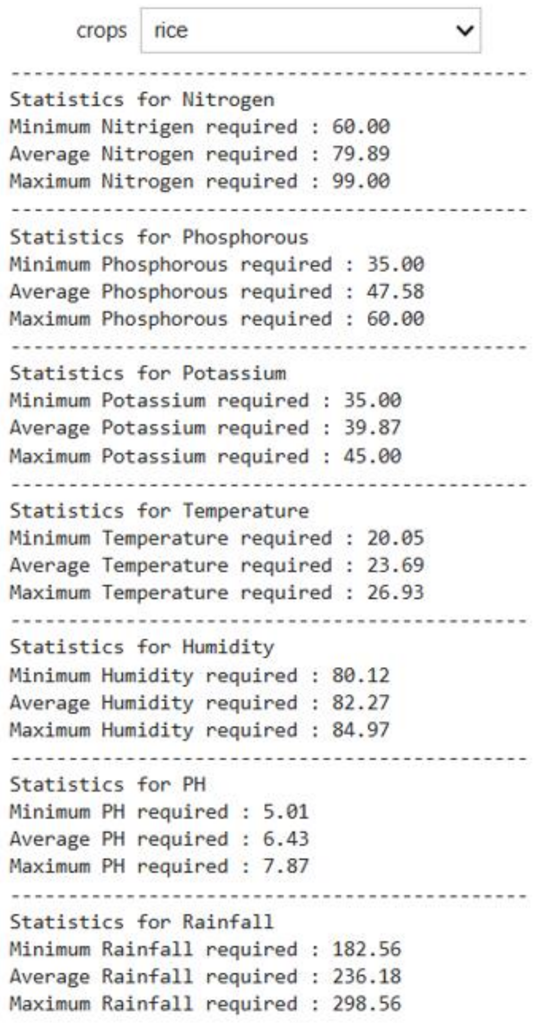

Density
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−25
0
25
50
75
100
125
150
N


Density
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250
K

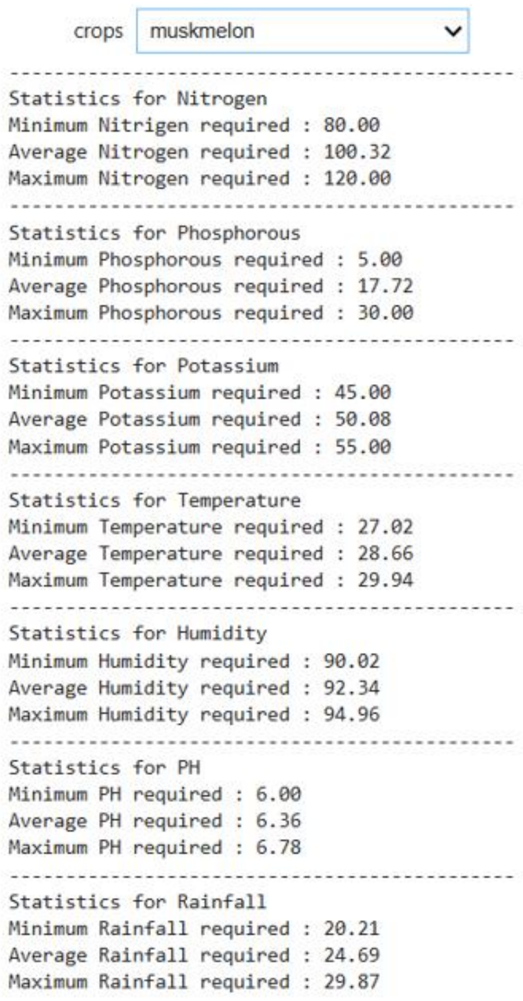

N
P
K
temperature
humidity
ph
rainfall
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

We can infer the Phosphorous levels and Potassium levels are highly correlated.

Deriving crop specific statistical parameters of the features



```
-------------------------------------------
Statistics for Nitrogen
Minimum Nitrigen required : 60.00
Average Nitrogen required : 79.89
Maximum Nitrogen required : 99.00
-------------------------------------------
Statistics for Phosphorous
Minimum Phosphorous required : 35.00
Average Phosphorous required : 47.58
Maximum Phosphorous required : 60.00
-------------------------------------------
Statistics for Potassium
Minimum Potassium required : 35.00
Average Potassium required : 39.87
Maximum Potassium required : 45.00
-------------------------------------------
Statistics for Temperature
Minimum Temperature required : 20.05
Average Temperature required : 23.69
Maximum Temperature required : 26.93
-------------------------------------------
Statistics for Humidity
Minimum Humidity required : 80.12
Average Humidity required : 82.27
Maximum Humidity required : 84.97
-------------------------------------------
Statistics for PH
Minimum PH required : 5.01
Average PH required : 6.43
Maximum PH required : 7.87
-------------------------------------------
Statistics for Rainfall
Minimum Rainfall required : 182.56
Average Rainfall required : 236.18
Maximum Rainfall required : 298.56
```

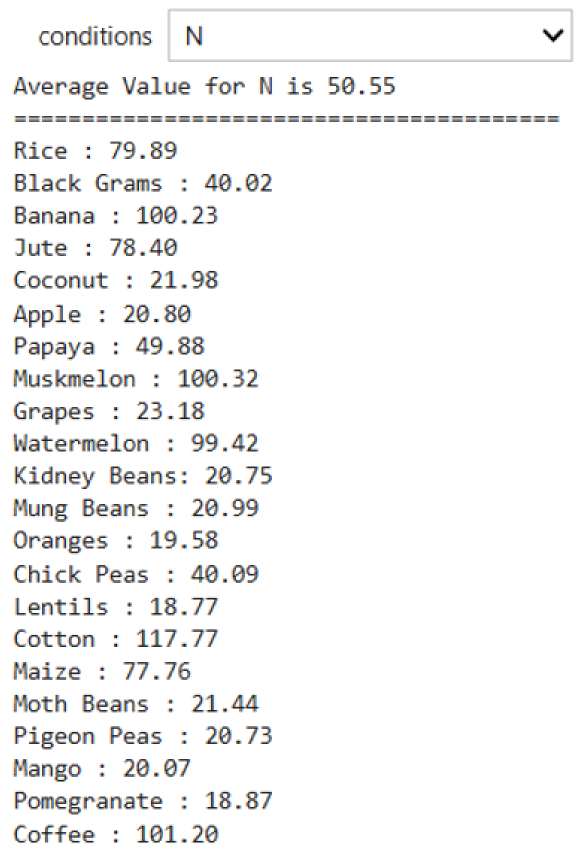


```
-------------------------------------------
Statistics for Nitrogen
Minimum Nitrigen required : 80.00
Average Nitrogen required : 100.32
Maximum Nitrogen required : 120.00
-------------------------------------------
Statistics for Phosphorous
Minimum Phosphorous required : 5.00
Average Phosphorous required : 17.72
Maximum Phosphorous required : 30.00
-------------------------------------------
Statistics for Potassium
Minimum Potassium required : 45.00
Average Potassium required : 50.08
Maximum Potassium required : 55.00
-------------------------------------------
Statistics for Temperature
Minimum Temperature required : 27.02
Average Temperature required : 28.66
Maximum Temperature required : 29.94
-------------------------------------------
Statistics for Humidity
Minimum Humidity required : 90.02
Average Humidity required : 92.34
Maximum Humidity required : 94.96
-------------------------------------------
Statistics for PH
Minimum PH required : 6.00
Average PH required : 6.36
Maximum PH required : 6.78
-------------------------------------------
Statistics for Rainfall
Minimum Rainfall required : 20.21
Average Rainfall required : 24.69
Maximum Rainfall required : 29.87
```

We can conclude that for rice plantation, on average rainfall should be above 236 mm. Similarly choose the land which has more amount of Nitrogen than Phosphorous and Potassium. Viewing muskmelon cultivation, we can infer that areas where rainfall is scare is suitable as this crop does not need much water intake. But at the same time, we need to ensure good humidity is obtained.

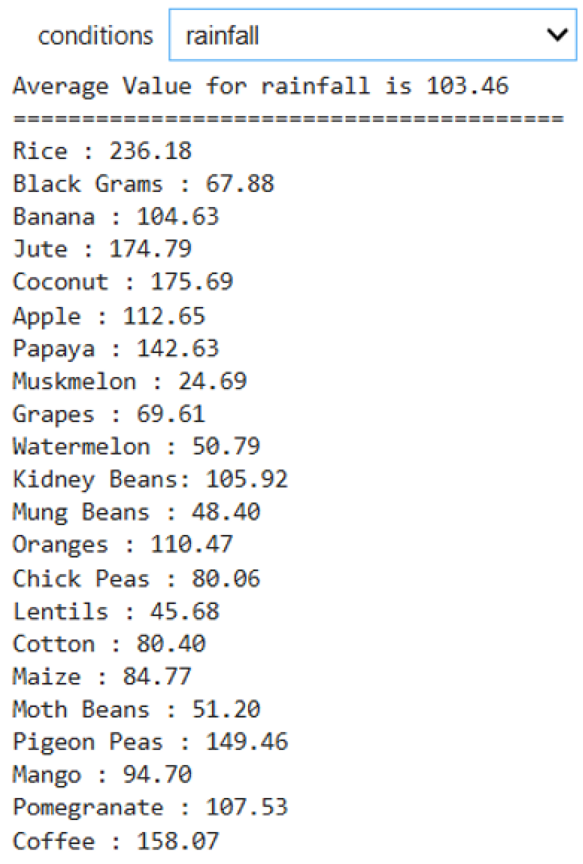
We developed an interactive widget by which, end user can select appropriate crop from the dropdown menu and get the Maximum, Minimum and Average values of the soil and climatic parameters.

Now, conversely for given parameters let us display the statistics for different crop types.



conditions: N

```
Average Value for N is 50.55
========================================
Rice : 79.89
Black Grams : 40.02
Banana : 100.23
Jute : 78.40
Coconut : 21.98
Apple : 20.80
Papaya : 49.88
Muskmelon : 100.32
Grapes : 23.18
Watermelon : 99.42
Kidney Beans: 20.75
Mung Beans : 20.99
Oranges : 19.58
Chick Peas : 40.09
Lentils : 18.77
Cotton : 117.77
Maize : 77.76
Moth Beans : 21.44
Pigeon Peas : 20.73
Mango : 20.07
Pomegranate : 18.87
Coffee : 101.20
```



conditions: rainfall

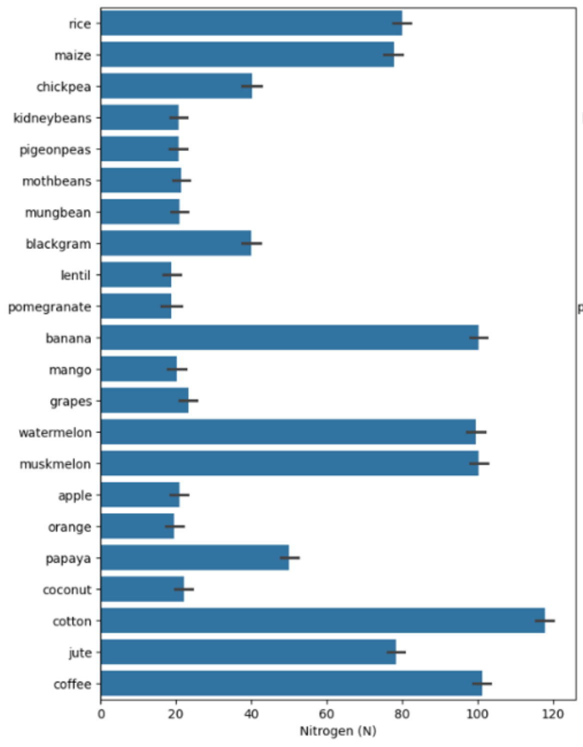
```
Average Value for rainfall is 103.46
========================================
Rice : 236.18
Black Grams : 67.88
Banana : 104.63
Jute : 174.79
Coconut : 175.69
Apple : 112.65
Papaya : 142.63
Muskmelon : 24.69
Grapes : 69.61
Watermelon : 50.79
Kidney Beans: 105.92
Mung Beans : 48.40
Oranges : 110.47
Chick Peas : 80.06
Lentils : 45.68
Cotton : 80.40
Maize : 84.77
Moth Beans : 51.20
Pigeon Peas : 149.46
Mango : 94.70
Pomegranate : 107.53
Coffee : 158.07
```

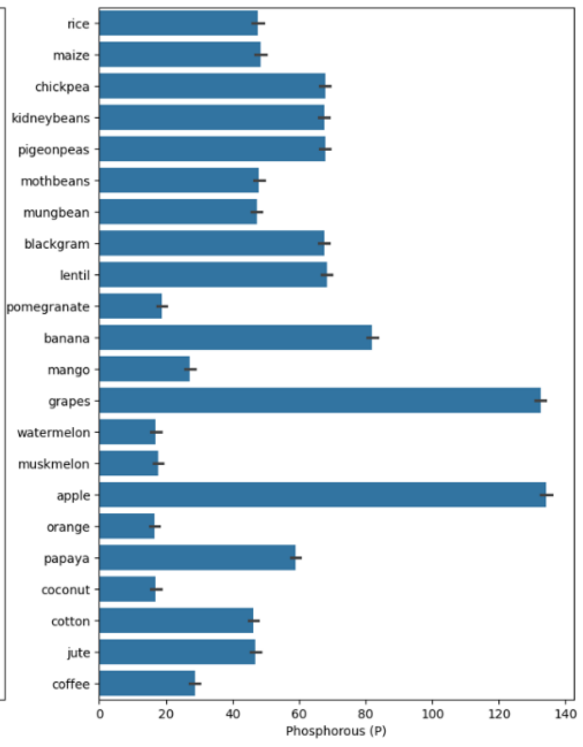
With this analysis, we can infer:

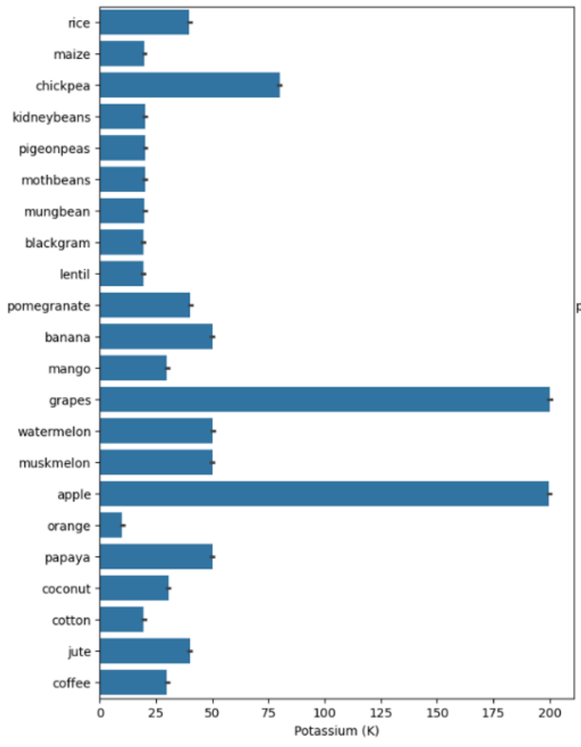
Soils with very less amount of Nitrogen (below average value i.e less than 50.55) can be utilized to cultivate Lentils, Oranges, Pomegranate, Beans, Coconuts. Whereas soils having very good amount of Nitrogen (greater than 50.55) can be utilized to cultivate Banana, Jute, Muskmelon, Watermelon, Cotton.

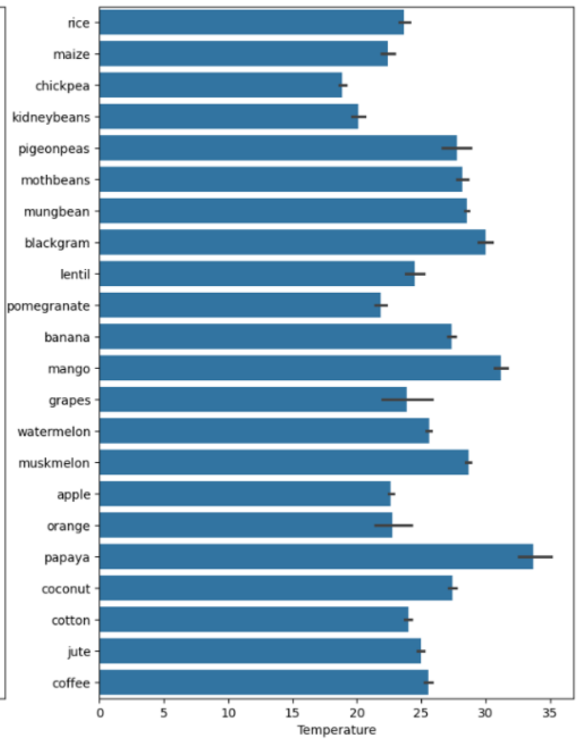
Coming to Rainfall metric, we can set the Average rainfall as 100 mm and conclude, areas having low rainfall are suggested to cultivate Lentils, Mung Beans, Muskmelon, Black Grams. Areas having good rainfall (greater than 100mm) are suggested to cultivate Rice, Pigeon Peas, Jute, Apple, Coconut, Papaya.

Similar analysis of viewing various crop's intake metrics to the soil and climate parameters done using bar charts.

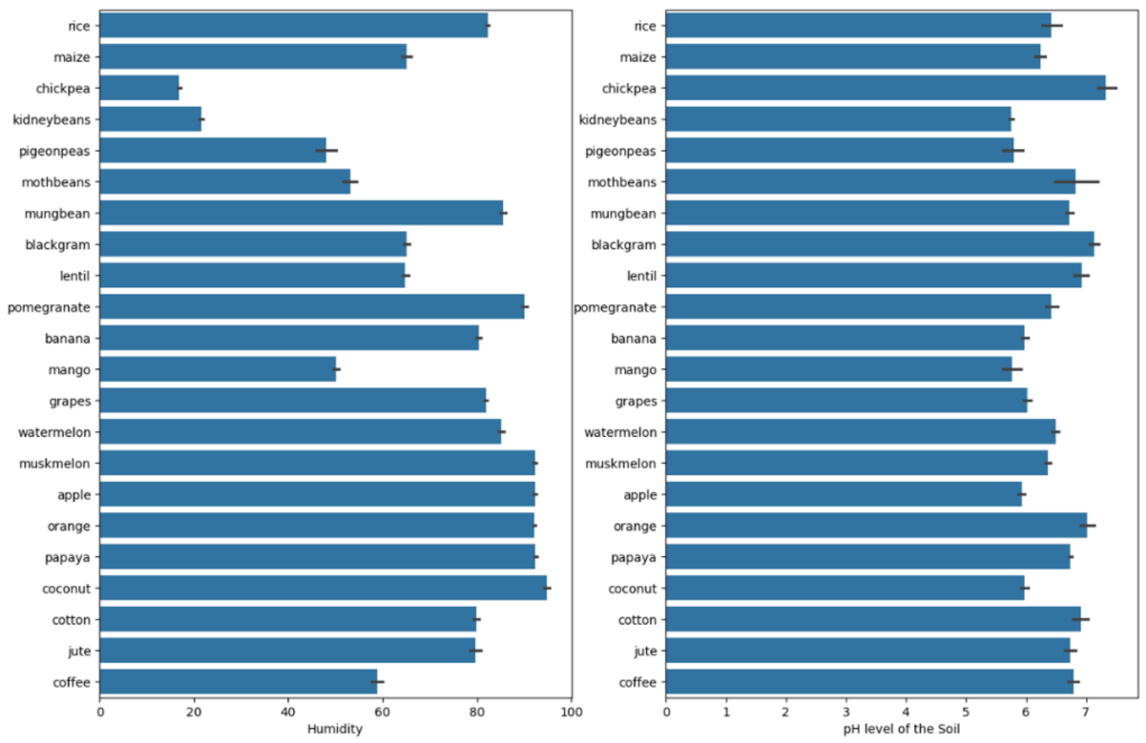


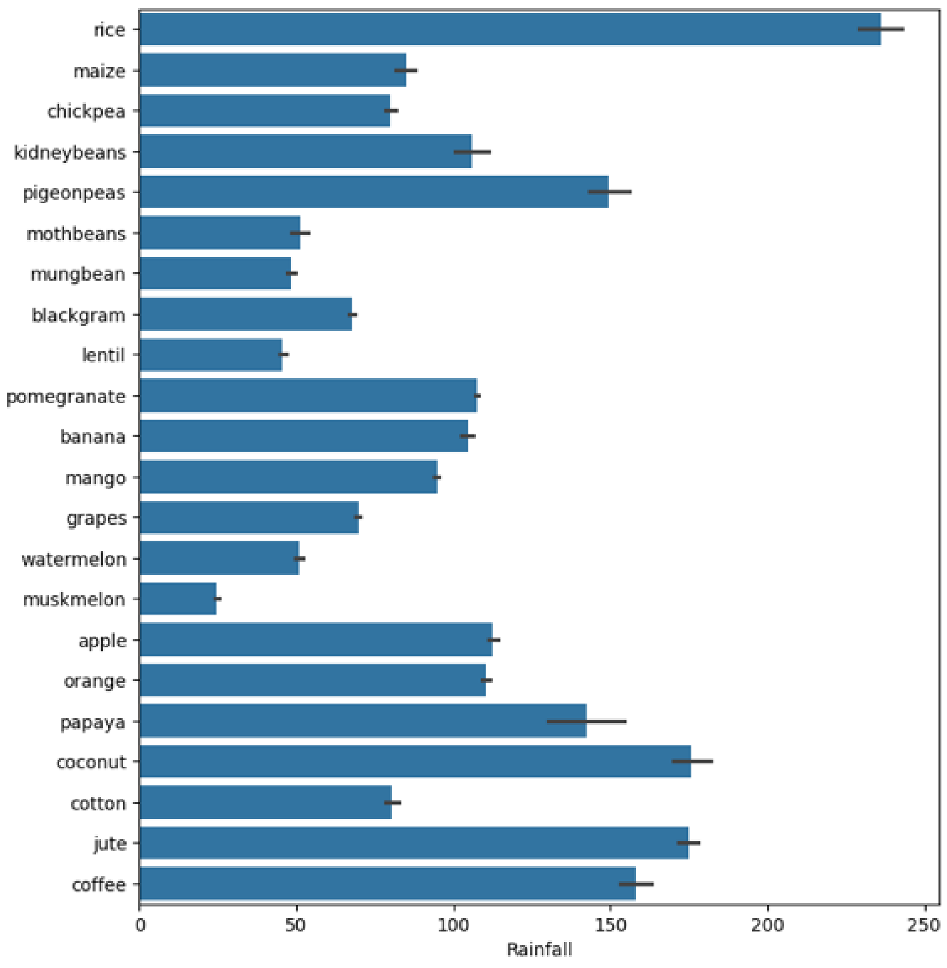






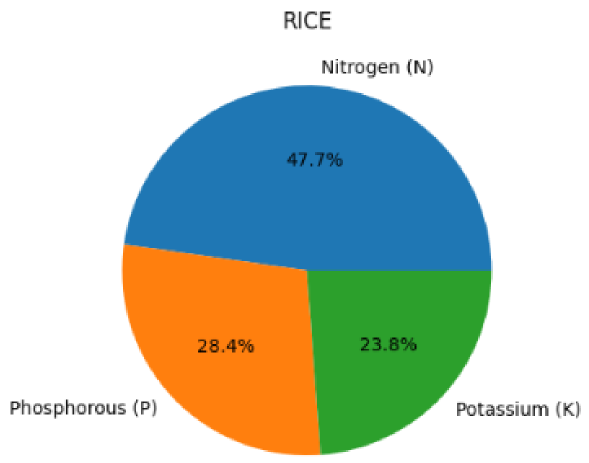
We can infer, Potassium levels are needed in high amount for only Grapes and Apple cultivations (Outliers). Rest all crops need minimal levels of Potassium.

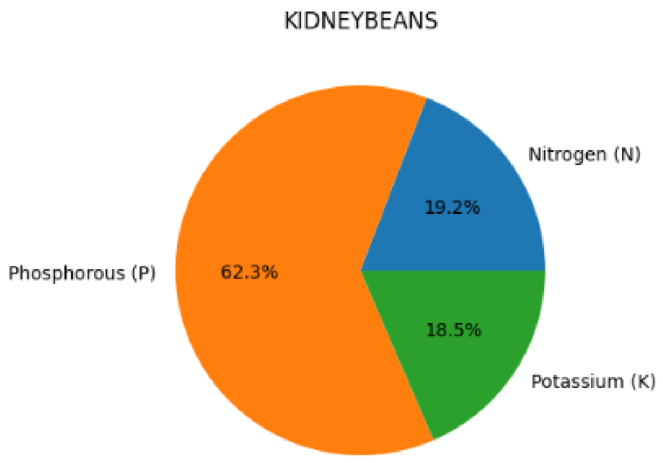

rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
Humidity
pH level of the Soil

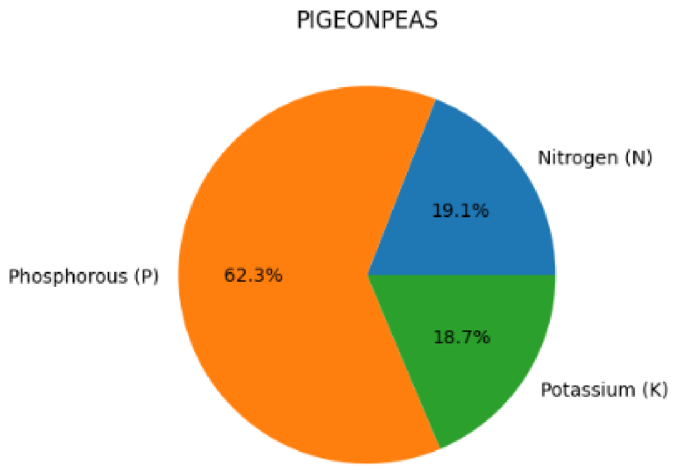

rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
Rainfall

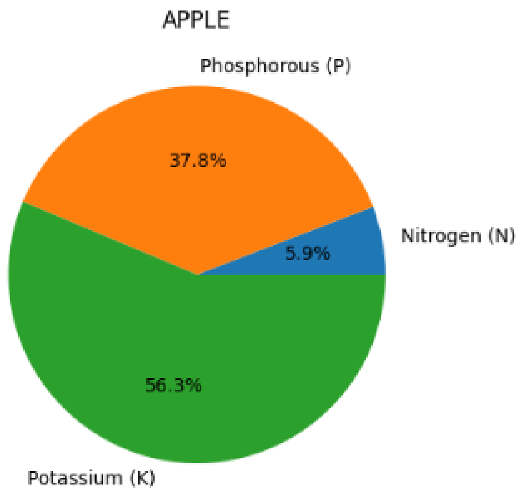
NPK RATIO

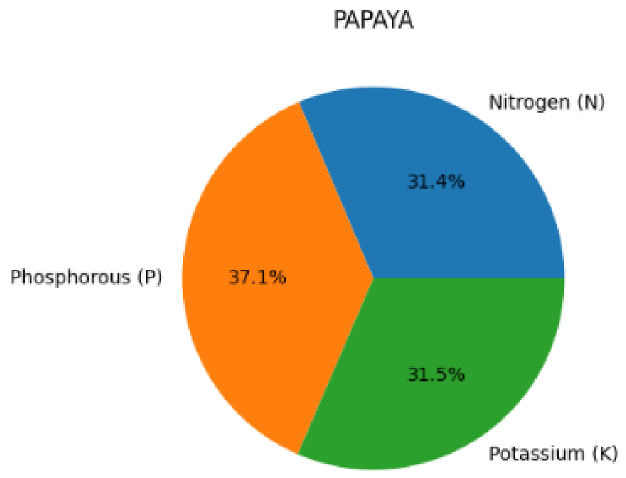
From the dataset, we grabbed top 6 crops and aggregated the means of Nitrogen (N), Phosphorous (P), Potassium (K) and plotted them on a Pie Chart.

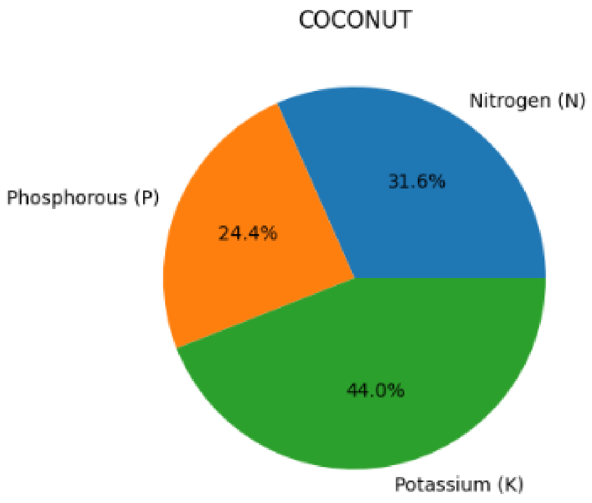




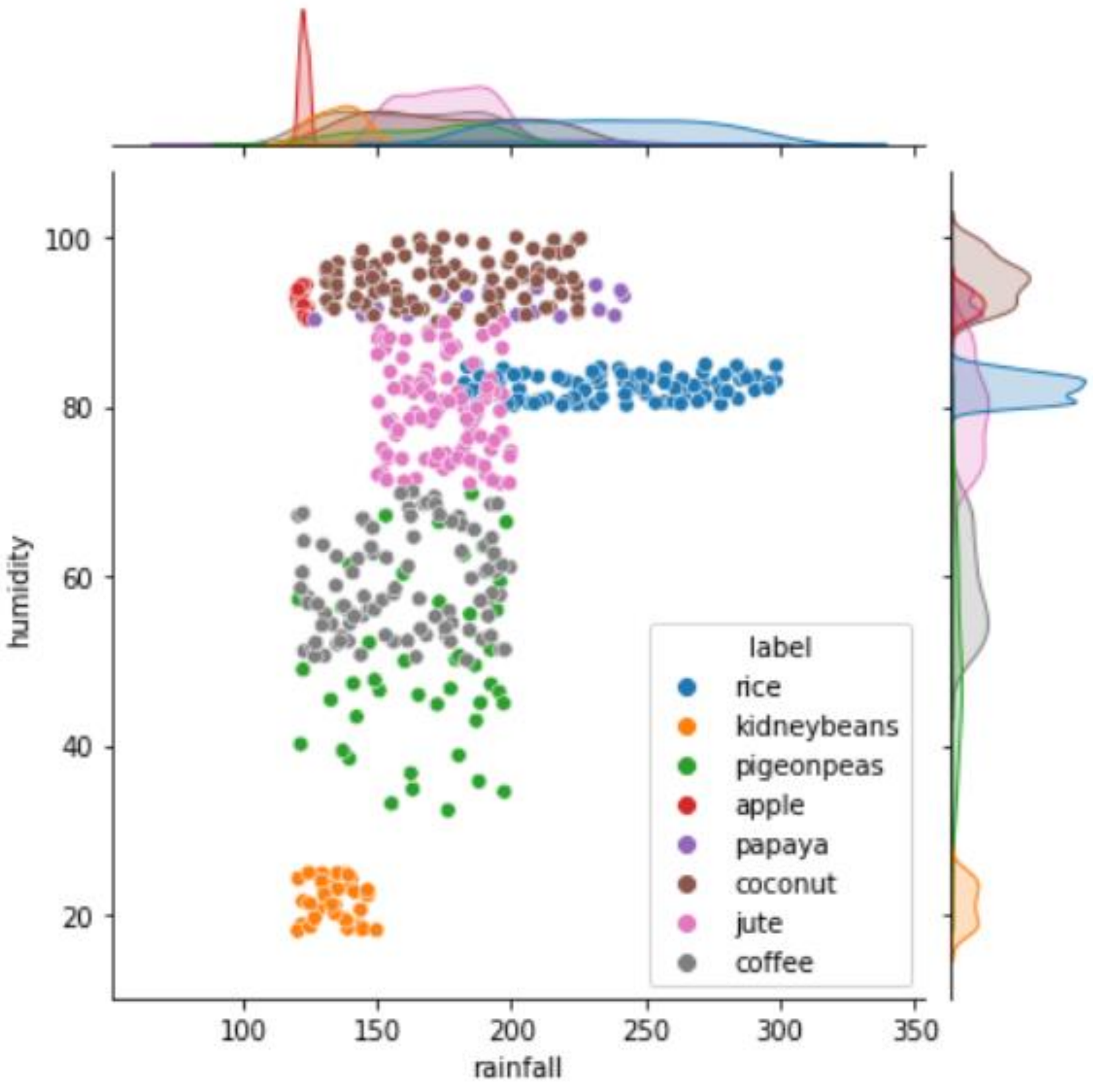








Now, let us focus on the Rainy Season where average rainfall is high (average 120 mm) and temperature is mildly chill (less than 30'C). Let us predict what crops are suitable during this season. Plotting a jointplot between humidity and rainfall.



We can infer from the above plot:

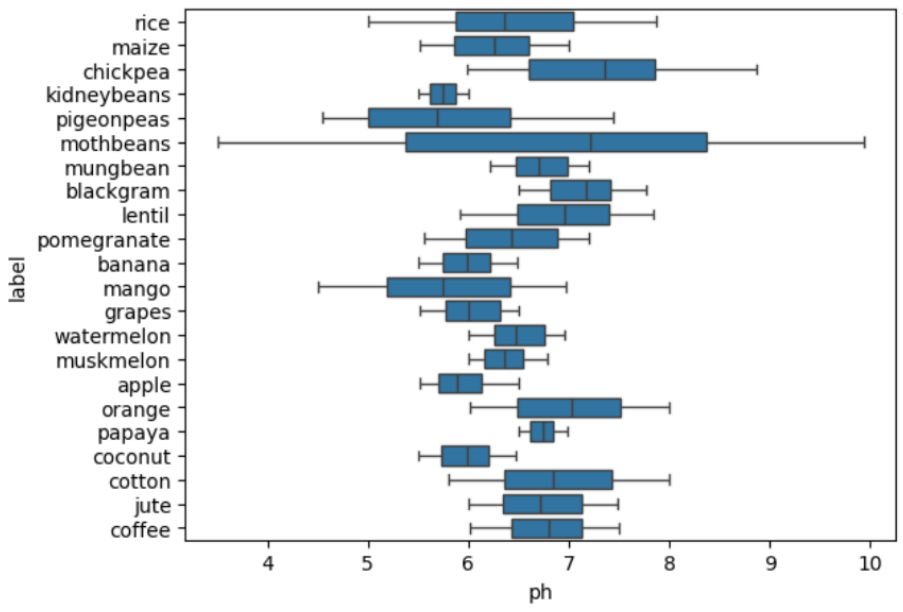
Rice production is suitable when rainfall is high and humid conditions are also greater than 80%.

Coconut needs high humidity conditions for the good yield, whereas kidneybeans require very less humid conditions.

Now, let us determine the range of pH needed for various crop cultivations

```
[8]: sns.boxplot(y='label',x='ph',data=df)
```

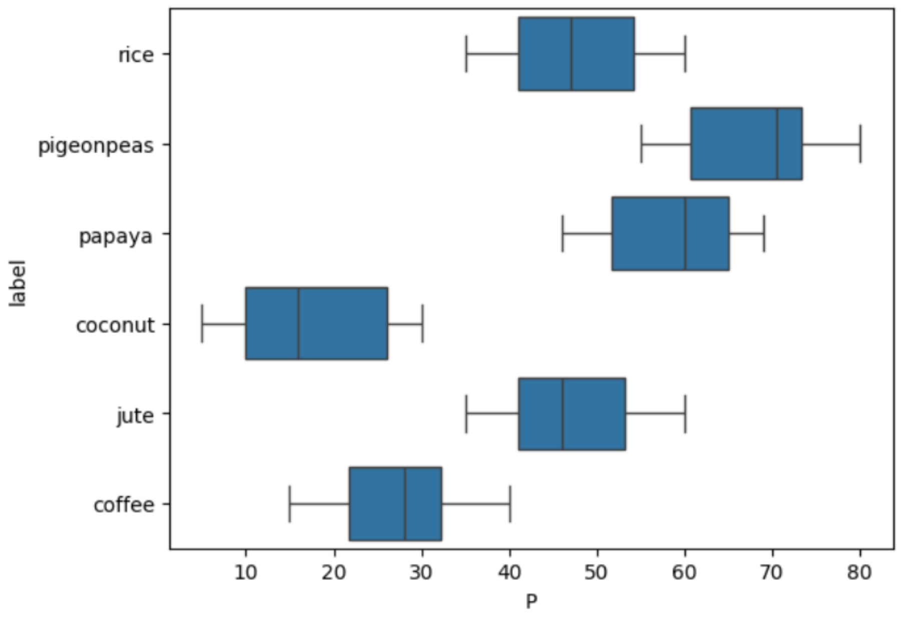
```
[8]: <Axes: xlabel='ph', ylabel='label'>
```



We can conclude that ph values are very critical when it comes to soil. A stability between 6 and 7 must be maintained for crop efficiency.

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

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



Similarly, we can infer that during Rainy Season, Phosphorous levels are quite differentiable.

MODEL IMPLEMENTATION

After exploring the data fields in the dataset, we will now try to train the model using popular machine learning algorithms and view the performance.

Classification using K Nearest Neighbours Classifier

```
#making X and Y components i.e Independent features/attributes and Target variable
c=df.label.astype('category')
trgts = dict(enumerate(c.cat.categories))
df['target'] = c.cat.codes

y=df.target
X = df[['N','P','K','temperature','humidity','ph','rainfall']]
```

```
trgts
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1)
```

```
#Performing Min Max Scaling to bring all the input features to the same scale
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
#k-NN classifier

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train_scaled,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

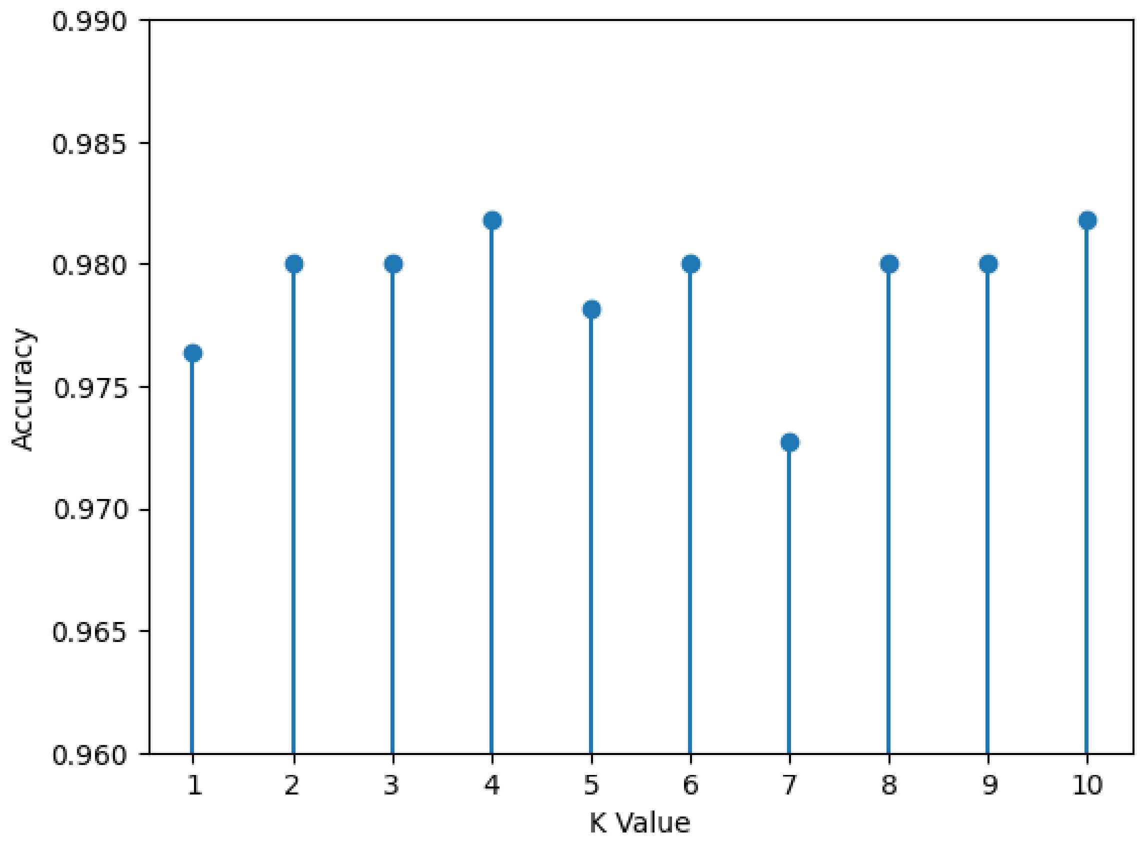
```
knn.score(X_test_scaled,y_test)
```

```
0.9781818181818182
```

We will now view, for different k values, how the model accuracy gets changed.

```
scores = []

for k in range(1,11):
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train_scaled,y_train)
    scores.append(knn.score(X_test_scaled,y_test))
```

```
k_range = range(1,11)
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.scatter(k_range, scores)
plt.vlines(k_range,0, scores, label = scores)
plt.ylim(0.96,0.99)
plt.xticks([i for i in range(1,11)]);
```



We can conclude that, for k = 4 we are getting maximum accuracy of 98.18%

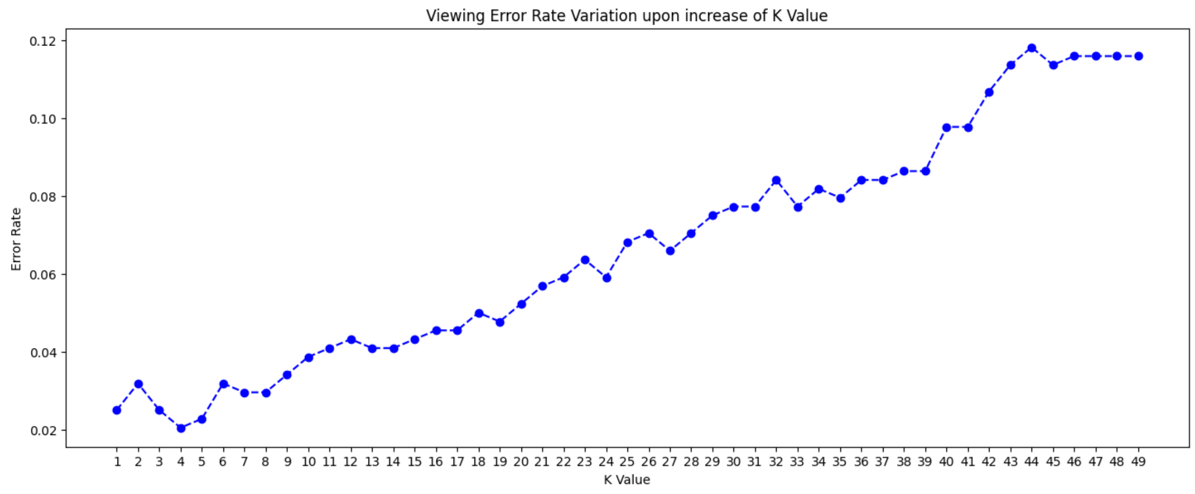
Now, let us view the error rate distribution for various k values:

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size = 0.2, random_state = 0)
print(f"Train Data: {X_train.shape}, {y_train.shape}")
print(f"Train Data: {X_test.shape}, {y_test.shape}")
```

```
Train Data: (1760, 7), (1760,)
Train Data: (440, 7), (440,)
```

```
#Performing multiple times k-NN with increasing K Value and examining the error rates

from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, confusion_matrix
error_rate = []
for i in range(1, 50):
    pipeline = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors = i))
    pipeline.fit(X_train, y_train)
    predictions = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    error_rate.append(np.mean(predictions != y_test))
```

```
plt.figure(figsize=(16,6))
plt.plot(range(1,50),error_rate,color='blue', linestyle='dashed', marker='o')
plt.title('Viewing Error Rate Variation upon increase of K Value')
plt.xlabel('K Value')
plt.xticks(range(1,50))
plt.ylabel('Error Rate')
```



We can infer:

Minimum Error is 0.0205 at K = 4

Maximum Error is 0.1182 at K = 44

Thus, from 2 plots we can confirm that optimal k value in k-NN is 4 (4 neighbours sufficient for classifying the data)

Classification using Support Vector Classifier (SVC)

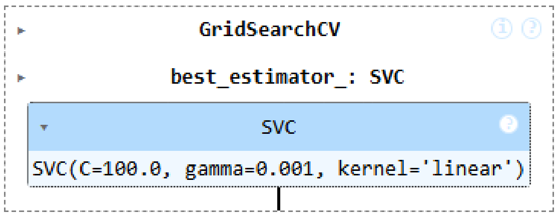
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1)
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svc_linear = SVC(kernel = 'linear').fit(X_train_scaled, y_train)
print("Linear Kernel Accuracy: ",svc_linear.score(X_test_scaled,y_test))

svc_rbf = SVC(kernel = 'rbf').fit(X_train_scaled, y_train)
print("RBF Kernel Accuracy: ", svc_rbf.score(X_test_scaled,y_test))

svc_poly = SVC(kernel = 'poly').fit(X_train_scaled, y_train)
print("Polynomial Kernel Accuracy: ", svc_poly.score(X_test_scaled,y_test))
```

```
Linear Kernel Accuracy:  0.9745454545454545
RBF Kernel Accuracy:  0.9872727272727273
Polynomial Kernel Accuracy:  0.9890909090909091
```

We can see, among the 3 types of kernels, using Polynomial Kernel yielded the maximum accuracy.

Now, let us perform **Hyperparameter Tuning using GridSearchCV**

```
from sklearn.model_selection import GridSearchCV

parameters = {'C': np.logspace(-3,2,6).tolist(), 'gamma': np.logspace(-3,2,6).tolist()}
model = GridSearchCV(estimator = SVC(kernel = 'linear'), param_grid=parameters, n_jobs = -1, cv = 4)
model.fit(X_train_scaled,y_train)
```



```
GridSearchCV
best_estimator_: SVC
SVC
SVC(C=100.0, gamma=0.001, kernel='linear')
```

```
print("Accuracy of Linear Kernel SVM before tuning is "+str(svc_linear.score(X_test_scaled,y_test)))

print("Accuracy of Linear Kernel SVM after tuning is "+str(model.best_score_ ))
```

```
Accuracy of Linear Kernel SVM before tuning is 0.9745454545454545
Accuracy of Linear Kernel SVM after tuning is 0.985457462563765
```

```
parameters = {'C': np.logspace(-4,5,7).tolist(), 'gamma': np.logspace(-4,5,7).tolist()}
model = GridSearchCV(estimator = SVC(kernel = 'rbf'), param_grid=parameters, n_jobs = -1, cv = 4)
model.fit(X_train_scaled,y_train)
```

```
GridSearchCV
best_estimator_: SVC
SVC
SVC(C=100.0, gamma=3.1622776601683795)
```

```
print("Accuracy of RBF Kernel SVM before tuning is "+str(svc_rbf.score(X_test_scaled,y_test)))

print("Accuracy of RBF Kernel SVM after tuning is "+str(model.best_score_ ))
```

```
Accuracy of RBF Kernel SVM before tuning is 0.9872727272727273
Accuracy of RBF Kernel SVM after tuning is 0.9909098121723594
```

```
parameters = {'C': np.logspace(-1,8,8).tolist(), 'gamma': np.logspace(-1,8,8).tolist()}
model = GridSearchCV(estimator = SVC(kernel = 'poly'), param_grid=parameters, n_jobs = -1, cv = 4)
model.fit(X_train_scaled,y_train)
```

```
GridSearchCV
best_estimator_: SVC
SVC
SVC(C=1.9306977288832505, gamma=1.9306977288832505, kernel='poly')
```

```
print("Accuracy of Polynomial Kernel SVM before tuning is "+str(svc_poly.score(X_test_scaled,y_test))

print("Accuracy of Polynomial Kernel SVM after tuning is "+str(model.best_score_ ))
```

```
Accuracy of Polynomial Kernel SVM before tuning is 0.9890909090909091
Accuracy of Polynomial Kernel SVM after tuning is 0.9878817085498014
```

We see, the results after tuning were better for both Linear kernel and RBF Kernel SVM, whereas for Polynomial Kernel SVM accuracy dropped after tuning. This might be for the reason that, already the initial set hyperparameters were the best ones for the given dataset to classify.

Classification using Decision Tree

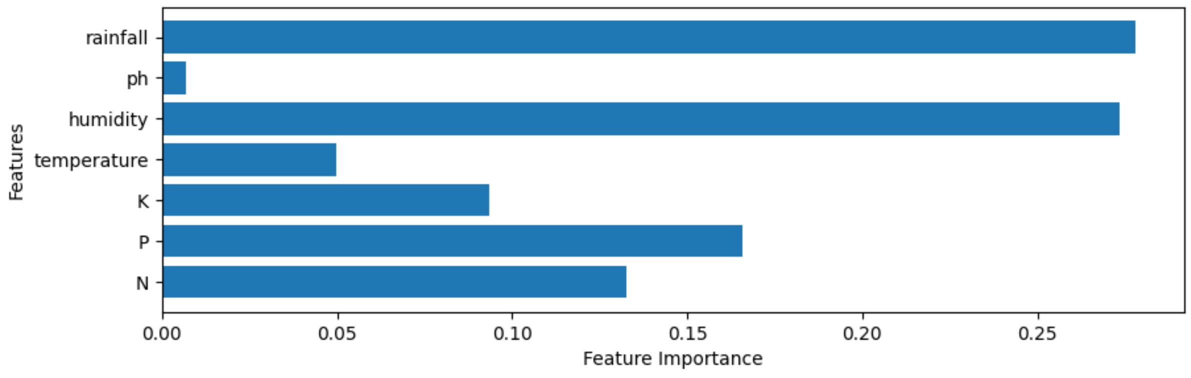
```
c=df.label.astype('category')
targets = dict(enumerate(c.cat.categories))
df['target']=c.cat.codes

y=df.target
X=df[['N','P','K','temperature','humidity','ph','rainfall']]
```

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=4)
```

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=42).fit(X_train,y_train)
print("Accuracy achieved in classifying the given data using Decision Tree is "+str(round(clf.score(X_test,y_test),4)))
```

```
Accuracy achieved in classifying the given data using Decision Tree is 0.9891
```

```
#Feature Importance used by Decision Tree Classifier
plt.figure(figsize=(10,3))
c_feat = len(X_train.columns)
plt.barh(range(c_feat),clf.feature_importances_)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.yticks(np.arange(c_feat),X_train.columns)
plt.show()
```



We can infer that both ‘rainfall’ and ‘humidity’ features were more impactful than the rest of the features during the Decision Tree classification.

Classification using Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
lf = RandomForestClassifier(max_depth=4,n_estimators=100,random_state=42).fit(X_train, y_train)
print('RF Accuracy on Testing Dataset: {:.2f}'.format(clf.score(X_test, y_test)))
```

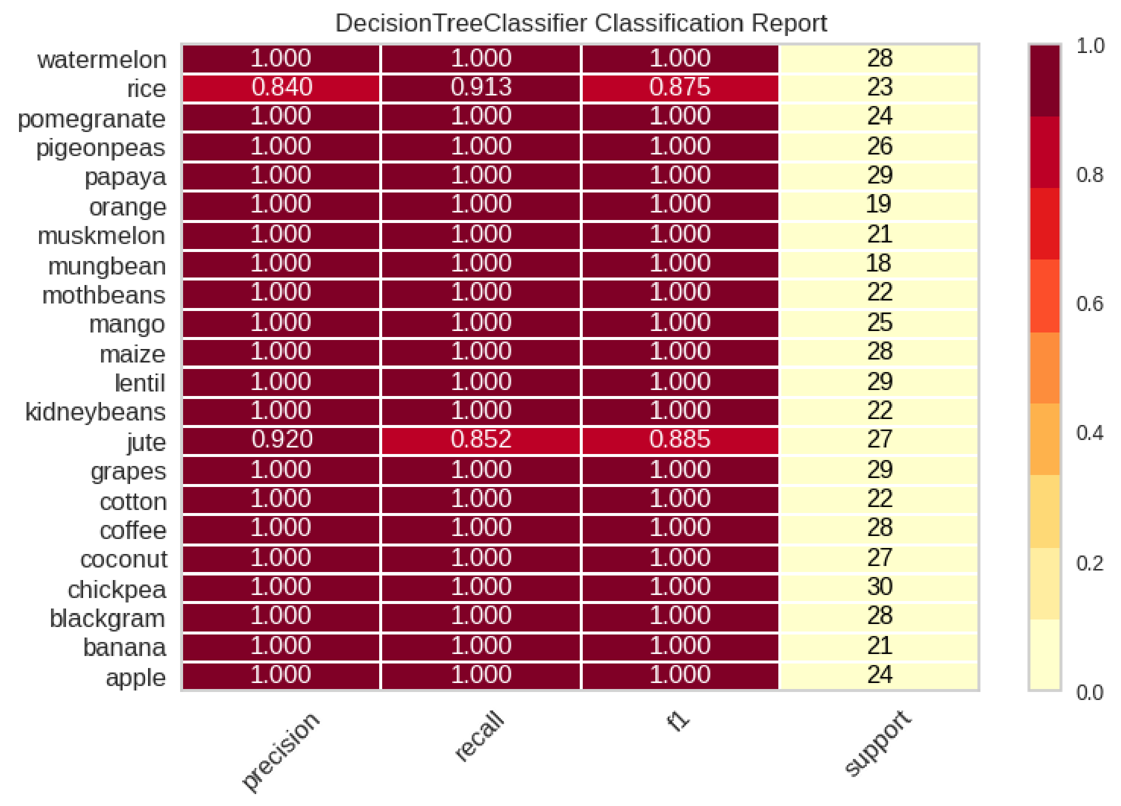
```
RF Accuracy on Testing Dataset: 0.99
```

It’s clear that Random Forest performed well as it is a balanced dataset.

We know that Random Forest algorithm makes the splits based on majority voting across trees. Hence in balanced dataset each class is represented equally, so the trees built in the forest don't overly favour any single class.

Let us now, display the Random Forest Classification Report which includes the values of the performance metrics to each target crop label.

```python
#Using yellowbrick to generate the RF Classification Report

from yellowbrick.classifier import ClassificationReport
classes = list(targets.values())
visualizer = ClassificationReport(clf, classes=classes, support=True)

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



	precision	recall	f1	support
watermelon	1.000	1.000	1.000	28
rice	0.840	0.913	0.875	23
pomegranate	1.000	1.000	1.000	24
pigeonpeas	1.000	1.000	1.000	26
papaya	1.000	1.000	1.000	29
orange	1.000	1.000	1.000	19
muskmelon	1.000	1.000	1.000	21
mungbean	1.000	1.000	1.000	18
mothbeans	1.000	1.000	1.000	22
mango	1.000	1.000	1.000	25
maize	1.000	1.000	1.000	28
lentil	1.000	1.000	1.000	29
kidneybeans	1.000	1.000	1.000	22
jute	0.920	0.852	0.885	27
grapes	1.000	1.000	1.000	29
cotton	1.000	1.000	1.000	22
coffee	1.000	1.000	1.000	28
coconut	1.000	1.000	1.000	27
chickpea	1.000	1.000	1.000	30
blackgram	1.000	1.000	1.000	28
banana	1.000	1.000	1.000	21
apple	1.000	1.000	1.000	24

We can see the classifier obtained Unity in most of the labels but didn't in few occurrences. In terms of precision measure, model didn't classify rice. Similarly, in terms of recall measure, model didn't classify jute. Thus, F1 scores are bad for jute and rice labels.

Making use of Light Gradient Boosting Machine (LightGBM) for depicting an example of Overfitting.

```
X = df.drop(['label','target'],axis=1)
y = df['label']
```

```
from sklearn.model_selection import train_test_split
import lightgbm as lgb
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,shuffle = True, random_state = 0)
mo = lgb.LGBMClassifier()
mo.fit(X_train,y_train)
```

```
#Checking whether model underwent Overfitting Problem

print('Training set score: {:.4f}'.format(mo.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(mo.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9894
```

We can see during training we achieved score of unity, but at the time of testing it was below the unity depicting the Model Overfitting case!

Now, let us view some real time predictions

```
#Giving real time parameters to the model to predict the suitable crop
# Values --> [N, P, K, Temp, Humidity, pH, Rainfall]
t1 = mo.predict([[90, 42, 43, 20.879744, 75, 5.5,220]])
t1
```

```
array(['rice'], dtype=object)
```

```
t2 = mo.predict([[50, 82, 23, 80.8, 25, 2.5,20]])
t2
```

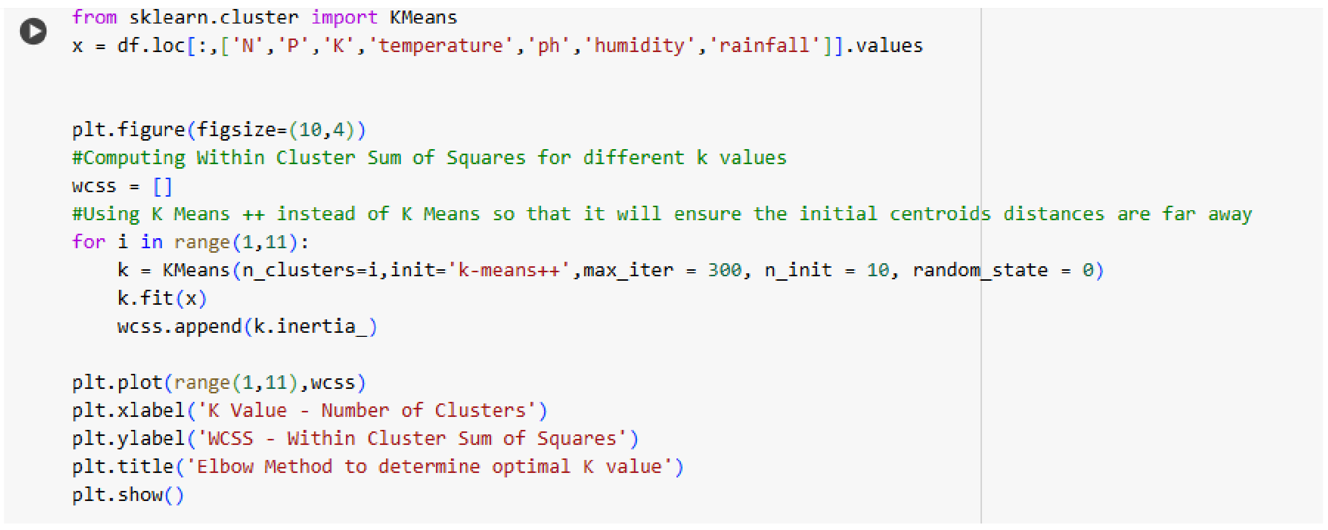
```
array(['muskmelon'], dtype=object)
```

```
t3 = mo.predict([[20,30,10,15,90,7.5,100]])
t3
```

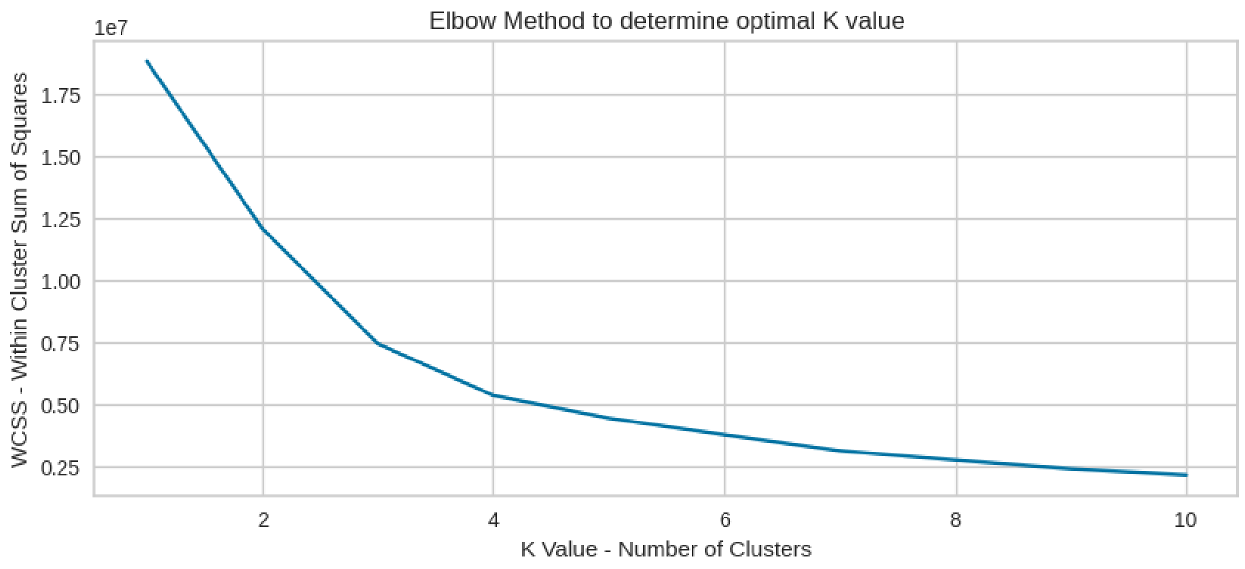
```
array(['orange'], dtype=object)
```

Note: These were the same results obtained when given to previous built models as well.

Classification using Clustering analysis

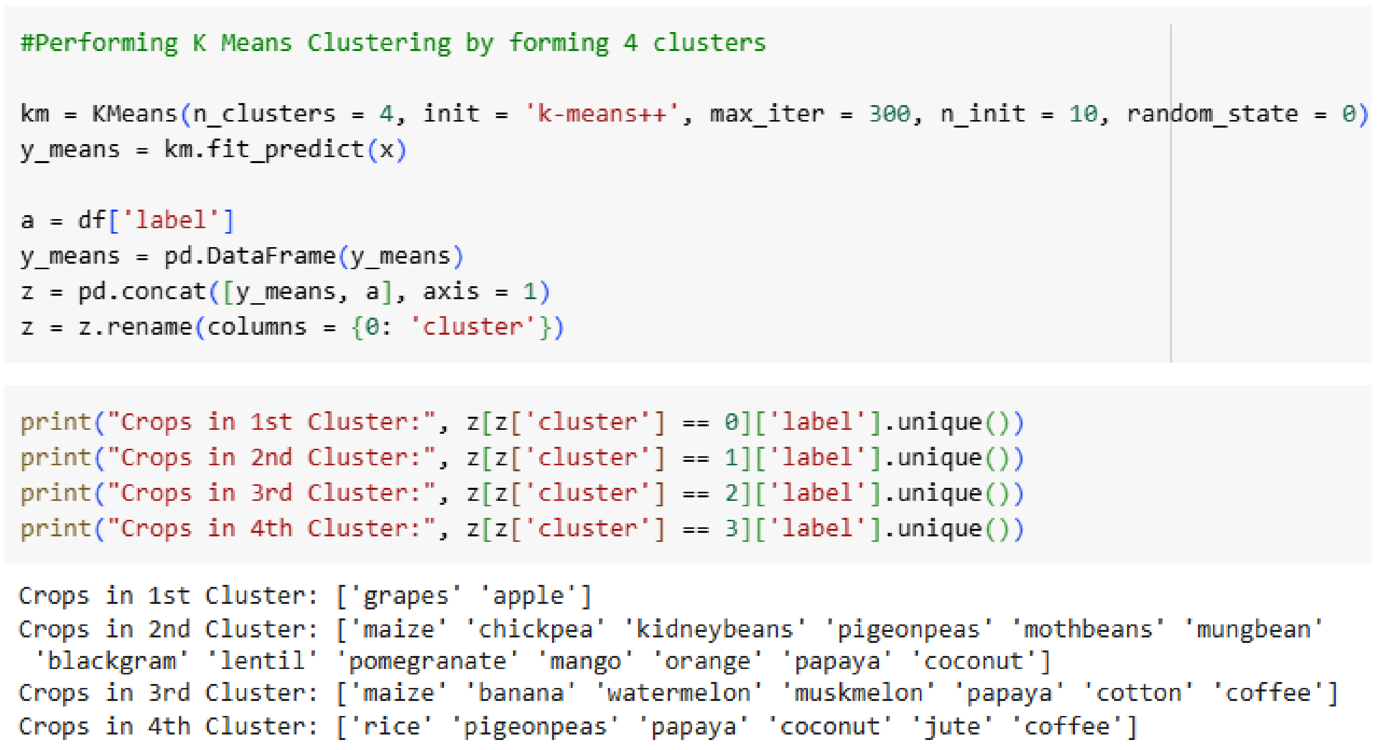


```
from sklearn.cluster import KMeans
x = df.loc[:,['N','P','K','temperature','ph','humidity','rainfall']].values


plt.figure(figsize=(10,4))
#Computing Within Cluster Sum of Squares for different k values
wcss = []
#Using K Means ++ instead of K Means so that it will ensure the initial centroids distances are far away
for i in range(1,11):
    k = KMeans(n_clusters=i,init='k-means++',max_iter = 300, n_init = 10, random_state = 0)
    k.fit(x)
    wcss.append(k.inertia_)

plt.plot(range(1,11),wcss)
plt.xlabel('K Value - Number of Clusters')
plt.ylabel('WCSS - Within Cluster Sum of Squares')
plt.title('Elbow Method to determine optimal K value')
plt.show()
```



Drawing the inference from Elbow Method, we can conclude that k=4 is the value at or near the elbow. Hence four is the reasonable number of clusters that adequately explains the structure of the data without unnecessary complexity.

So, let us see how the model classifies the labels into 4 clusters. We make use of (k Means ++) instead of ordinary (k Means) because in k Means ++, model will ensure the initial centroids of the clusters are having large amount of distances between them.

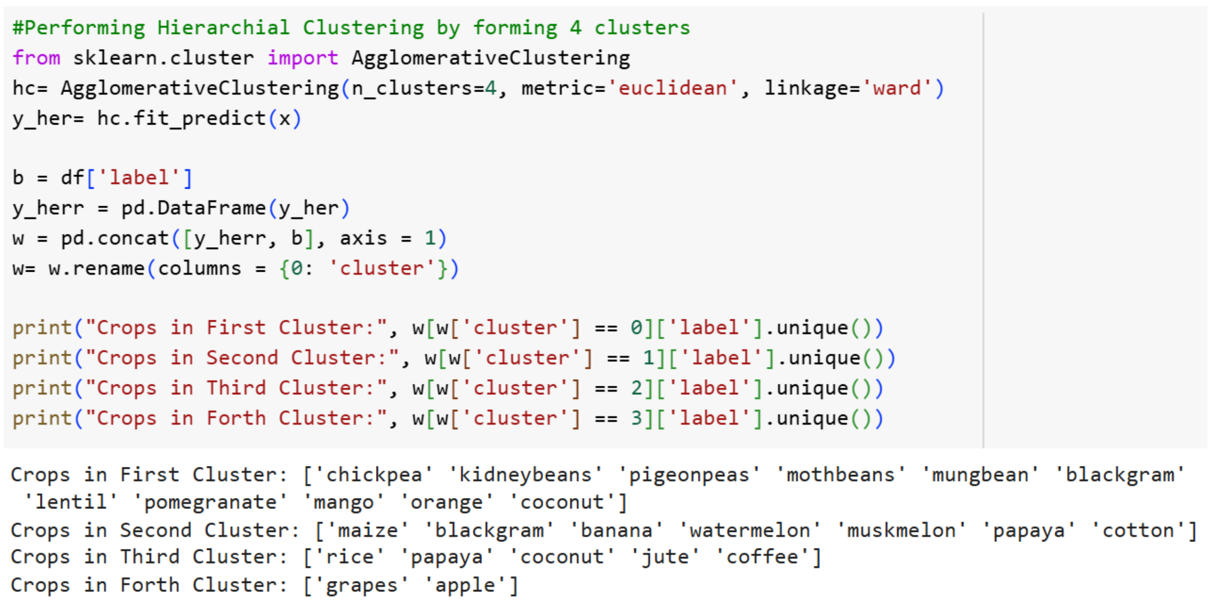


```
#Performing K Means Clustering by forming 4 clusters

km = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_means = km.fit_predict(x)

a = df['label']
y_means = pd.DataFrame(y_means)
z = pd.concat([y_means, a], axis = 1)
z = z.rename(columns = {0: 'cluster'})
```

```
print("Crops in 1st Cluster:", z[z['cluster'] == 0]['label'].unique())
print("Crops in 2nd Cluster:", z[z['cluster'] == 1]['label'].unique())
print("Crops in 3rd Cluster:", z[z['cluster'] == 2]['label'].unique())
print("Crops in 4th Cluster:", z[z['cluster'] == 3]['label'].unique())
```

```
Crops in 1st Cluster: ['grapes' 'apple']
Crops in 2nd Cluster: ['maize' 'chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans' 'mungbean'
 'blackgram' 'lentil' 'pomegranate' 'mango' 'orange' 'papaya' 'coconut']
Crops in 3rd Cluster: ['maize' 'banana' 'watermelon' 'muskmelon' 'papaya' 'cotton' 'coffee']
Crops in 4th Cluster: ['rice' 'pigeonpeas' 'papaya' 'coconut' 'jute' 'coffee']
```

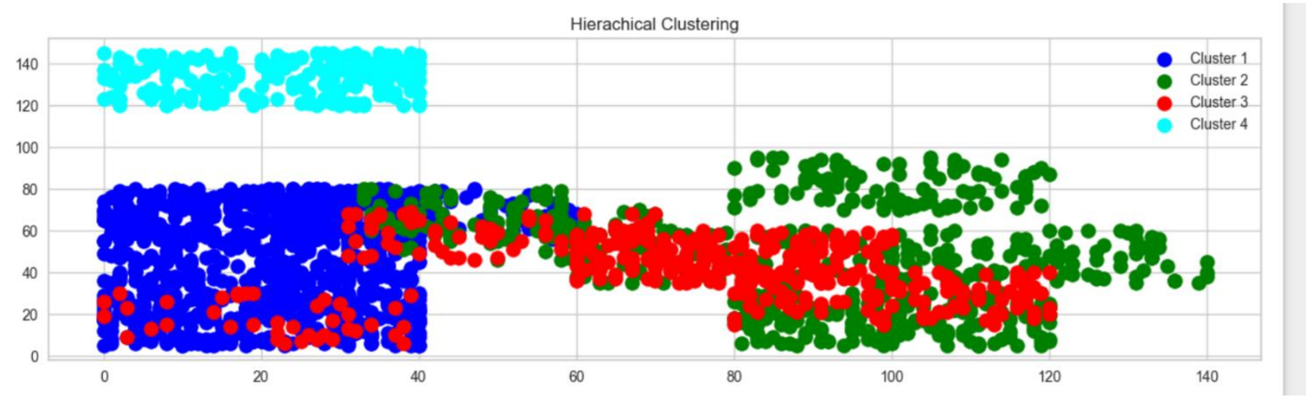
Classification using Hierarchical Clustering Analysis



```
#Performing Hierarchial Clustering by forming 4 clusters
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=4, metric='euclidean', linkage='ward')
y_her= hc.fit_predict(x)

b = df['label']
y_herr = pd.DataFrame(y_her)
w = pd.concat([y_herr, b], axis = 1)
w= w.rename(columns = {0: 'cluster'})

print("Crops in First Cluster:", w[w['cluster'] == 0]['label'].unique())
print("Crops in Second Cluster:", w[w['cluster'] == 1]['label'].unique())
print("Crops in Third Cluster:", w[w['cluster'] == 2]['label'].unique())
print("Crops in Forth Cluster:", w[w['cluster'] == 3]['label'].unique())
```

```
Crops in First Cluster: ['chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans' 'mungbean' 'blackgram'
 'lentil' 'pomegranate' 'mango' 'orange' 'coconut']
Crops in Second Cluster: ['maize' 'blackgram' 'banana' 'watermelon' 'muskmelon' 'papaya' 'cotton']
Crops in Third Cluster: ['rice' 'papaya' 'coconut' 'jute' 'coffee']
Crops in Forth Cluster: ['grapes' 'apple']
```

We see almost same clusters formed when implemented by two different clustering models. In Hierarchical Clustering, we have taken Euclidean Distance as a metric in forming the clusters.


Hierachical Clustering
Cluster 1
Cluster 2
Cluster 3
Cluster 4
0
20
40
60
80
100
120
140

Coming to the Secondary Dataset (Crop Yield Prediction Dataset)

Data from FAO (Food and Agriculture Organization)

pesticides.csv – [Domain, Area, Element, Item, Year, Unit, Value]

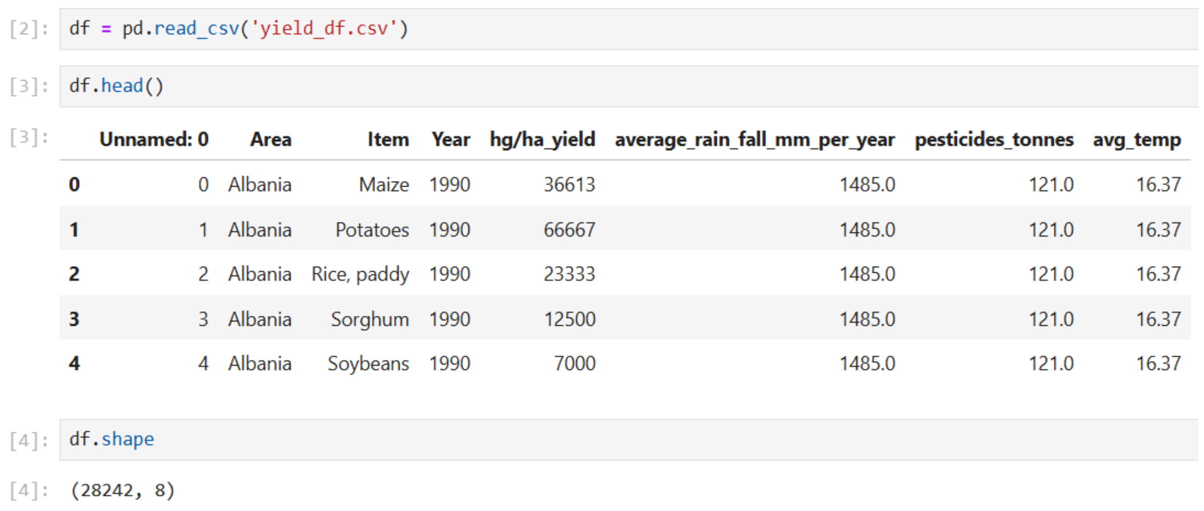
yield.csv – [Domain Code, Domain, Area Code, Area, Element Code, Element, Item Code, Item, Year Code, Year]

Data from World Data Bank

rainfall.csv – [Area, Year, average_rain_fall_mm_per_year]

temp.csv – [Year, Country, avg_temp]

yield_df.csv is final dataset merging of pesticides, yield, rainfall & avg. temp.



```
[2]: df = pd.read_csv('yield_df.csv')
```

```
[3]: df.head()
```

[3]:

	Unnamed: 0	Area	Item	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
0	0	Albania	Maize	1990	36613	1485.0	121.0	16.37
1	1	Albania	Potatoes	1990	66667	1485.0	121.0	16.37
2	2	Albania	Rice, paddy	1990	23333	1485.0	121.0	16.37
3	3	Albania	Sorghum	1990	12500	1485.0	121.0	16.37
4	4	Albania	Soybeans	1990	7000	1485.0	121.0	16.37

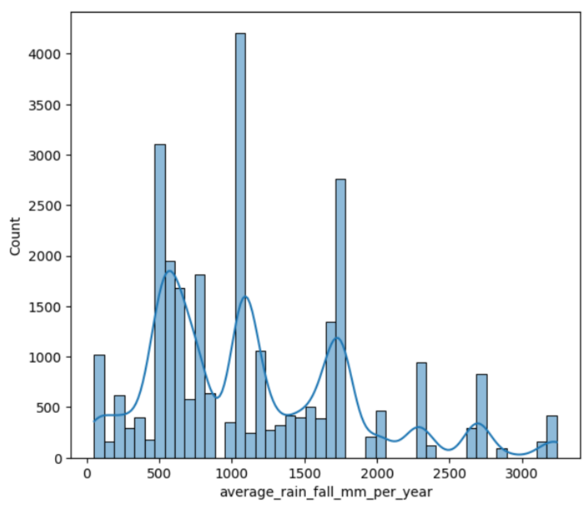
```
[4]: df.shape
```

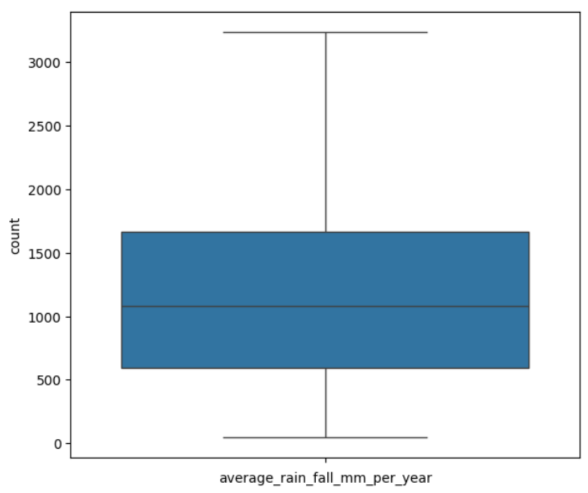
```
[4]: (28242, 8)
```

- **Area :** It describe the area in which the crops grows
- **Item :** It indicate the name of the crops
- **Year :** It indicate the year of crops grow
- **average_rain_fall_mm_per_year :** This variable indicate the average rain fall in mm per year
- **pesticides_tonnes :** It indicate how much pesticides uses in tonnes
- **avg_temp :** It indicate the Average temprature of area

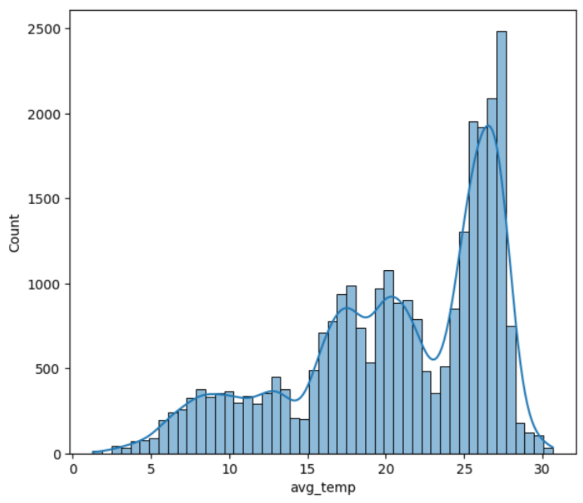
Data Exploration

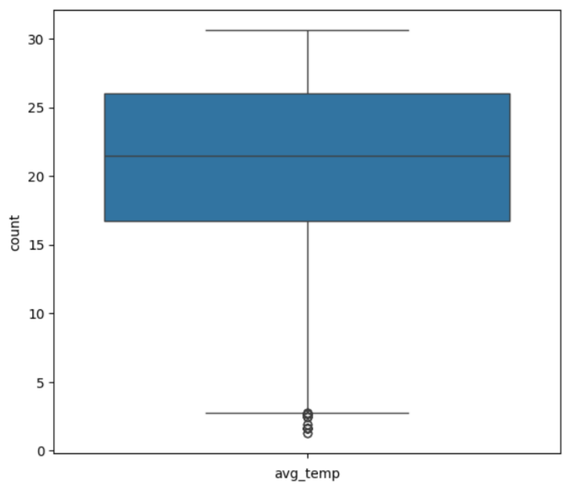
Let us view how the data attributes distributed over the dataset. Initial focus would be on the 3 vital features namely Average Rainfall (mm per year), Average Temperature and Amount of Pesticides used in tonnes.



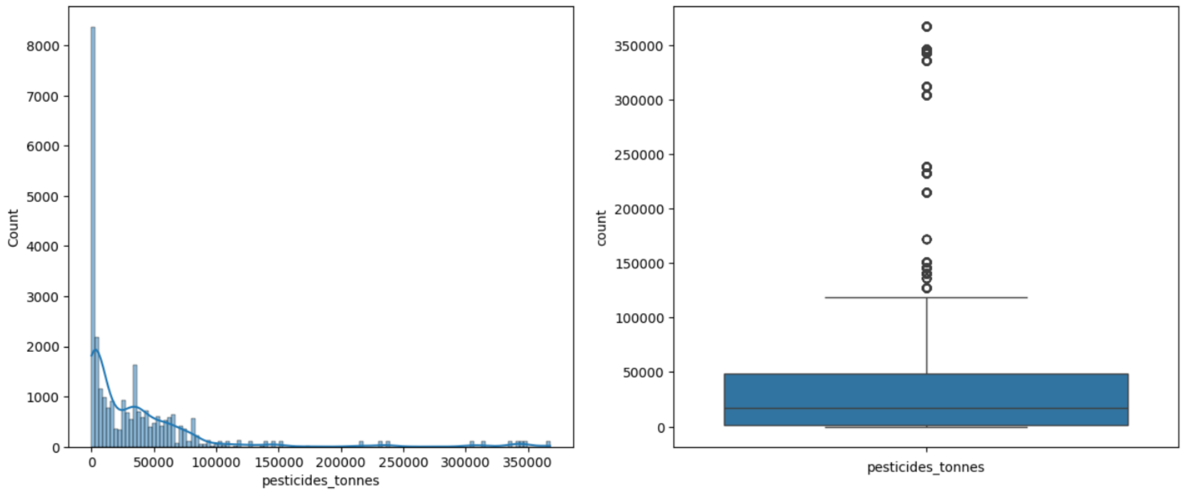


We can conclude, average rainfall feature can be considered as normally distributed and do not contain outliers.





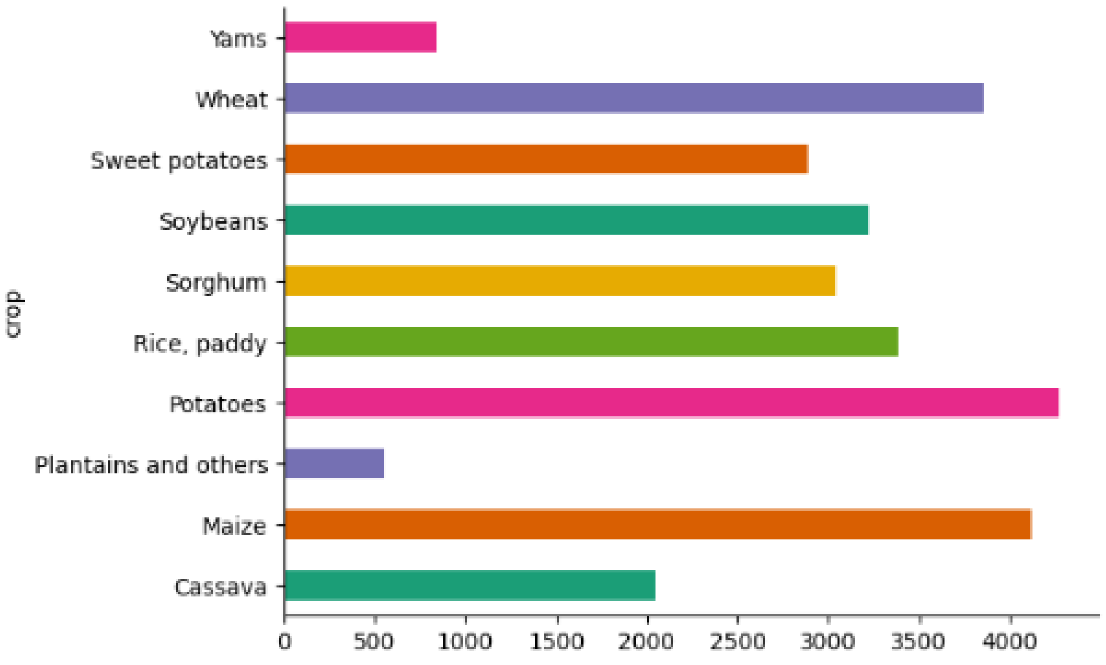
We can conclude, average temperature feature is also normally distributed and contain very few outliers.



We can infer that pesticides_tonnes feature is right skewed and contains many outliers. Hence, special attention to be taken during Data Preprocessing to transform the data and make it normally distributed.

Checking for Class Balance in the dataset

```python
from matplotlib import pyplot as plt
import seaborn as sns
df.groupby('crop').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Checking for Correlation

```
numerical_columns = ['Year', 'yield', 'average_rain_fall_mm_per_year', 'pesticides_tonnes', 'avg_temp']

# the correlation matrix
correlation = df[numerical_columns].corr()

print(correlation['yield'].sort_values(ascending=False))

# Plotting the correlation matrix as a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f", square=True, cbar_kws={"shrink": .8})
plt.title('Correlation Matrix for Numerical Columns')
plt.show()
```

```
yield                            1.000000
Year                             0.091630
pesticides_tonnes                0.064085
average_rain_fall_mm_per_year    0.000962
avg_temp                        -0.114777
```

Insights drawn:

hg/ha_yield : This is the correlation of the target variable with itself and is always Unity.

Year : There is a small positive correlation between the year and the crop yield (hg/ha_yield). This suggests that crop yield slightly increases over time, but the relationship is weak.

Pesticides_tonnes : There is a very weak positive correlation between the amount of pesticides used and the crop yield. This indicates that an increase in pesticide usage has a very slight association with an increase in yield, but it's not a strong relationship.

average_rain_fall_mm_per_year : This correlation is nearly zero, meaning there's no noticeable relationship between rainfall and crop yield in this data. Rainfall doesn't seem to impact the yield based on this correlation significantly.

avg_temp : This is a weak negative correlation between average temperature and crop yield. It suggests that as the temperature increases the crop yield slightly decreases.

MODELS IMPLEMENTATION

Data Preprocessing

As mentioned earlier, pesticides_tonnes feature was right skewed, hence need to perform “Boxcox Transformation” to make it near to normal distribution.

```
from scipy.stats import boxcox

boxcox(df["pesticides_tonnes"])
df["avg_boxcox"], param = boxcox(df["pesticides_tonnes"])
df["avg_boxcox"].skew()
```

```
-0.14159023174487217
```

```
df["pesticides_tonnes"] = df["avg_boxcox"]
df.drop(columns=["avg_boxcox"], inplace=True)
```

Performing Label Encoding for the “Area” and “Crop” features

```
from sklearn.preprocessing import LabelEncoder

label_en = LabelEncoder()
df["Area"] = label_en.fit_transform(df["Area"])
df["crop"] = label_en.fit_transform(df["crop"])
```

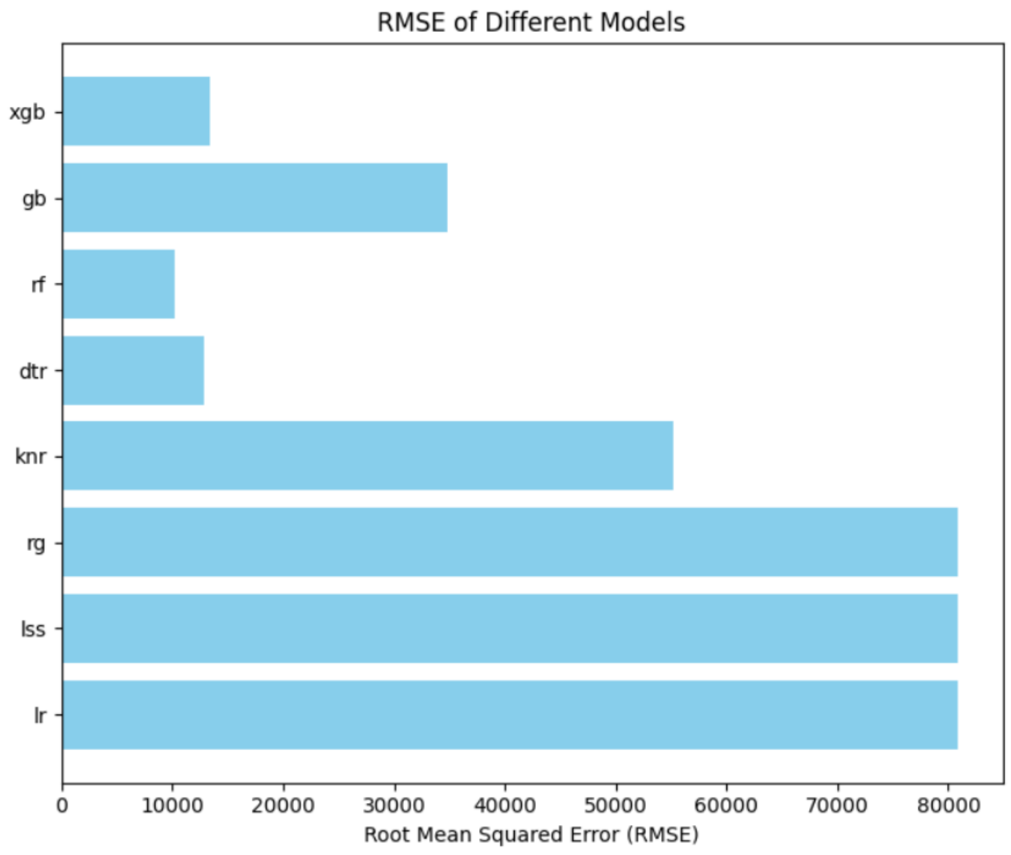
We try to plot the variations shown by each machine learning algorithm towards the Root Mean Squared Error (RMSE) and R^2 Score.

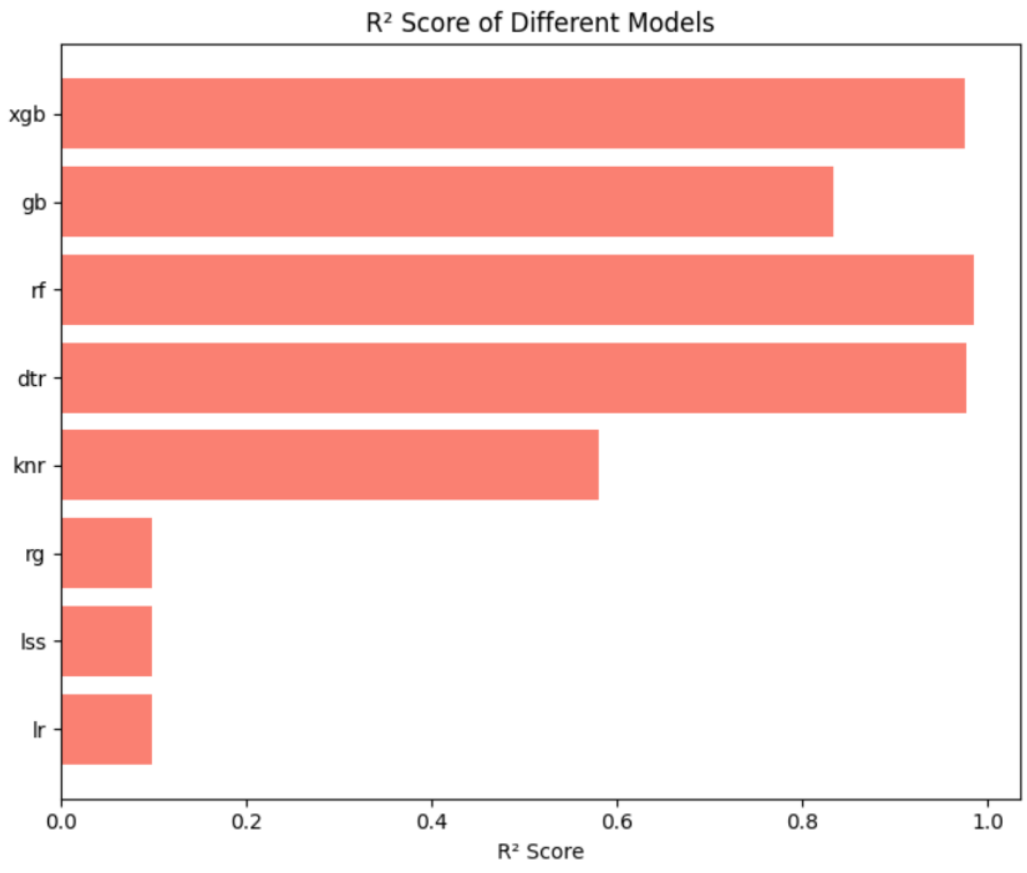
Abbreviations used in the plotting are:

lr = Linear Regression, lss = Lasso Regression, rg = Ridge Regression

knr = K Neighbors Regression dtr = Decision Tree Regression

rf = Random Forest gb = Gradient Boosting xgb = Extreme Gradient Boosting


RMSE of Different Models
xgb
gb
rf
dtr
knr
rg
lss
lr
0
10000
20000
30000
40000
50000
60000
70000
80000
Root Mean Squared Error (RMSE)


R² Score of Different Models
xgb
gb
rf
dtr
knr
rg
lss
lr
0.0
0.2
0.4
0.6
0.8
1.0
R² Score

Conclusions:

Decision Tree Regressor (DTR): Model achieves the lowest Mean Squared Error (MSE) and a high R^2 score, suggesting it performs excellently in terms of both prediction accuracy and model fit. It demonstrates strong capabilities in predicting the target variable with minimal error.

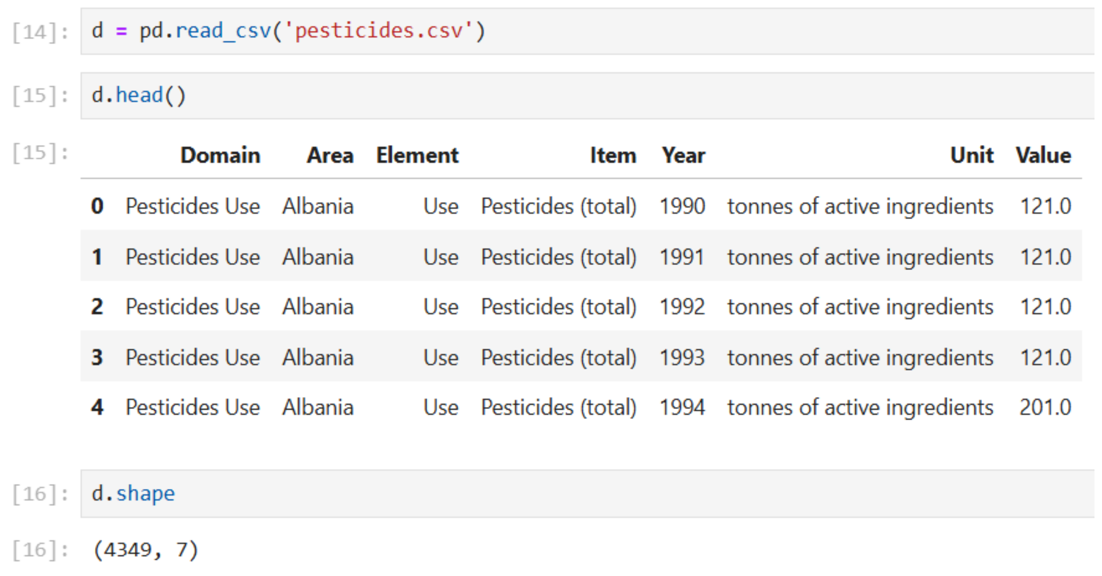
Random Forest Regressor (RF): The Random Forest Regressor has the lowest MSE of all the models, signifying it has the smallest average squared difference between predicted and actual values. It also boasts the highest R^2 score, making it the most accurate model overall.

XGBoost Regressor (XGB): Model performs well with a low MSE and a high R^2 score, though it falls slightly behind the Random Forest Regressor in terms of accuracy and fit.

Therefore, the best model is **Random Forest Regressor (RF)** stands out with the lowest MSE and the highest R^2 score.

Pesticides Dataset

pesticides.csv – [Domain, Area, Element, Item, Year, Unit, Value]



```
[14]: d = pd.read_csv('pesticides.csv')
```

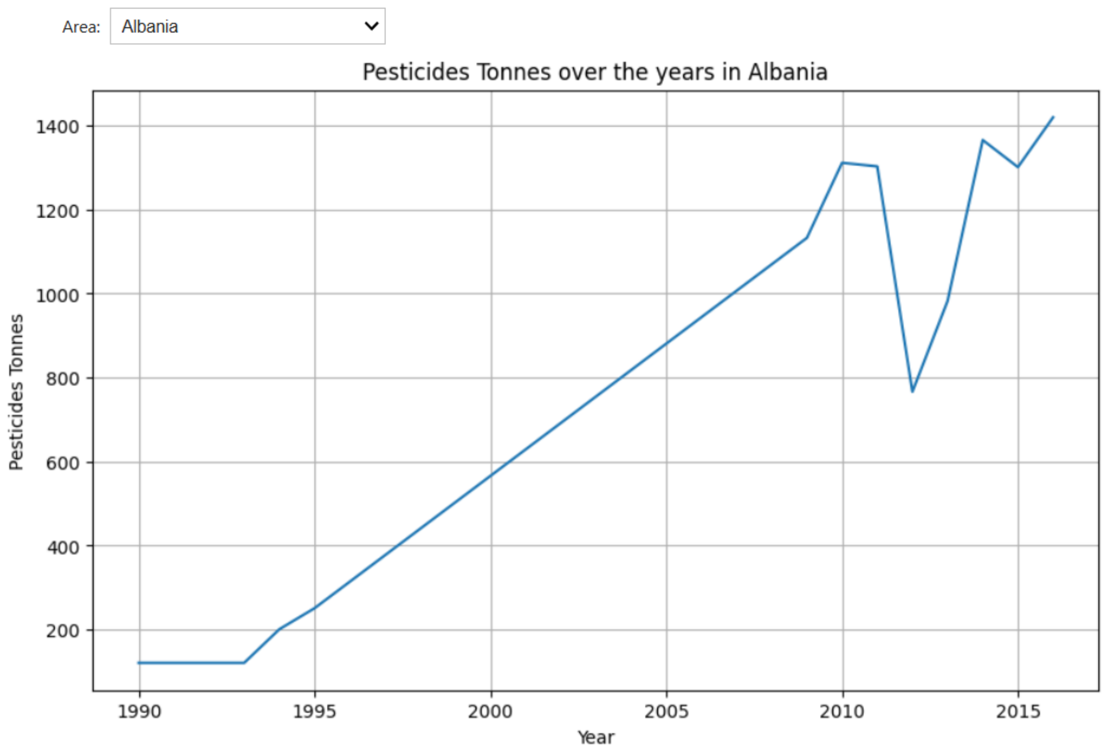
```
[15]: d.head()
```

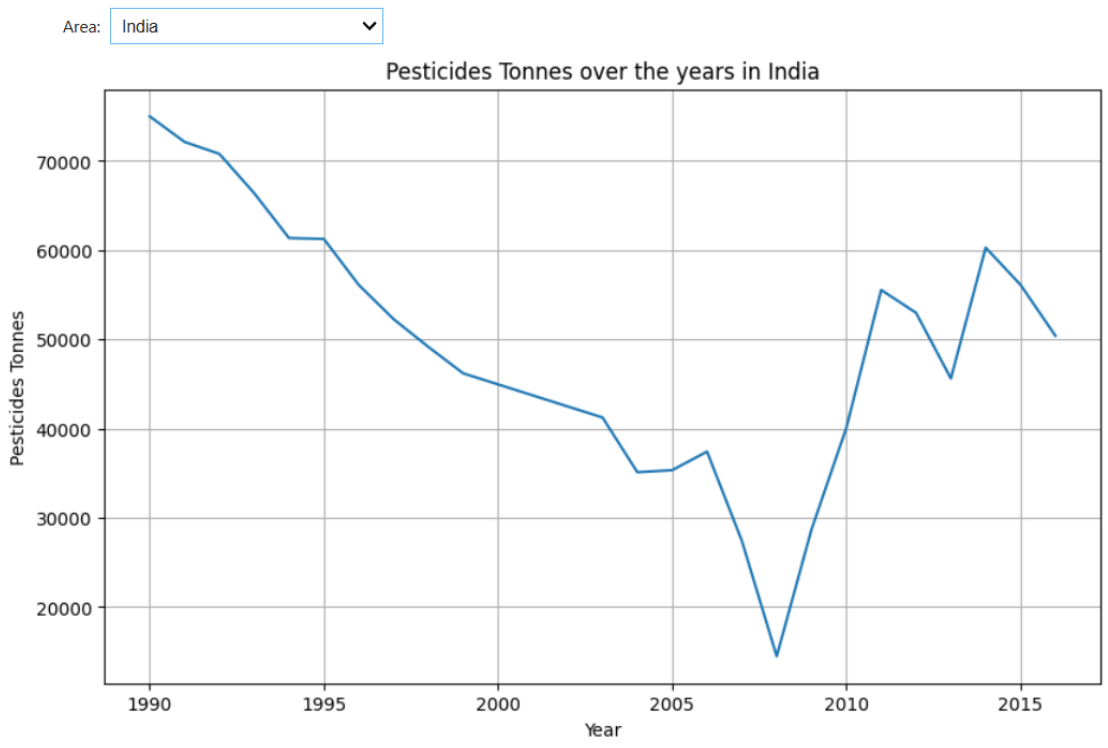
[15]:

	Domain	Area	Element	Item	Year	Unit	Value
0	Pesticides Use	Albania	Use	Pesticides (total)	1990	tonnes of active ingredients	121.0
1	Pesticides Use	Albania	Use	Pesticides (total)	1991	tonnes of active ingredients	121.0
2	Pesticides Use	Albania	Use	Pesticides (total)	1992	tonnes of active ingredients	121.0
3	Pesticides Use	Albania	Use	Pesticides (total)	1993	tonnes of active ingredients	121.0
4	Pesticides Use	Albania	Use	Pesticides (total)	1994	tonnes of active ingredients	201.0

```
[16]: d.shape

[16]: (4349, 7)
```

Data Visualization





We developed interactive plot so that, end user can select which ever country he wants and can view the distribution of amount of pesticides used over the past years.

Data Preprocessing (Performing One Hot Encoding)

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```python
X = d.drop('Value', axis=1)
y = d['Value']
categorical_features = ['Domain','Area', 'Element','Item']
numerical_features = ['Year']
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

```python
encoder = OneHotEncoder(handle_unknown='ignore')
X_train_encoded = encoder.fit_transform(X_train[categorical_features])
X_test_encoded = encoder.transform(X_test[categorical_features])

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[numerical_features])
X_test_scaled = scaler.transform(X_test[numerical_features])

import scipy.sparse as sp
import numpy as np
X_train_preprocessed = sp.hstack((X_train_scaled, X_train_encoded))
X_test_preprocessed = sp.hstack((X_test_scaled, X_test_encoded))
```

Classification using Decision Tree Regressor

[22]:
```python
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(random_state=101)
model.fit(X_train_preprocessed, y_train)
```

[22]:
```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=101)
```

[23]:
```python
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
y_pred = model.predict(X_test_preprocessed)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print('Decision Tree:')
print(f'Mean Squared Error: {mse:.3f}')
print(f'Mean Absolute Error: {mae:.3f}')
print(f'R^2 Score: {r2:.3f}')
```

```
Decision Tree:
Mean Squared Error: 58057818.623
Mean Absolute Error: 1513.595
R^2 Score: 0.996
```

Project Implementation

[Source Code] available in Git Repository.

Future Works

1) Perform LLM taking agriculture data of each state – region specific analysis.
2) Depict the mathematical inferences and derivations to the conclusions given by models.
3) Deploy the efficient model and make it available to farmers with responsive UI

Acknowledgement

We sincerely thank our Course Instructor, Dr Swapnil Vishveshwar Hingmire for giving us the opportunity to implement the learning outcomes of the course in a practical manner. Accomplishing the project, enhanced our skills and expertise on Machine Learning. Learnt various algorithms and model building. Got to know about efficient EDA techniques and mechanisms.

****