Agriculture- Project on Machine Learning

* **Objective:**

In this blogger, I am going to explain various machine learning model on the famous Agriculture dataset, to explore multiple predictions and will save the best model. Through analysis, we will also determine the outcome of the harvest season, i.e. whether the crop would be healthy (alive),damaged by pesticides or damaged by other reasons.

A tractor in a field

Description automatically generated with low confidence

* **Let’s talk about Agriculture.**

**Agriculture** is the art and science of cultivating the soil, growing crops and raising livestock. It includes the preparation of plant and animal products for people to use and their distribution to markets. **Agriculture** provides most of the world's food and fabrics.

A picture containing outdoor, person, sky, tree

Description automatically generated

Recently we have observed the emerging concept of smart farming that makes agriculture more efficient and effective with the help of high-precision algorithms. The mechanism that drives it is Machine Learning — the scientific field that gives machines the ability to learn without being strictly programmed. It has emerged together with big data technologies and high-performance computing to create new opportunities to unravel, quantify, and understand data intensive processes in agricultural operational environments.

**A picture containing text, grass, sign

Description automatically generated**

**Machine learning** is everywhere throughout the whole growing and harvesting cycle. It begins with a seed being planted in the soil — from the soil preparation, seeds breeding and water feed measurement — and it ends when neural networks pick up the harvest determining the ripeness with the help of computer vision.

# The Toxic Pesticides

Though, many of us don't appreciate much, but a farmer's job is real test of endurance and determination. Once the seeds are sown, he works days and nights to make sure that he cultivates a good harvest at the end of season. A good harvest is ensured by several factors such as availability of water, soil fertility, protecting crops from rodents, timely use of pesticides & other useful chemicals and nature. While a lot of these factors are difficult to control for, the amount and frequency of pesticides is something the farmer can control.

A picture containing grass, outdoor, plant

Description automatically generated

Pesticides are also special, because while they protect the crop with the right dosage. But, if you add more than required, they may spoil the entire harvest. A high level of pesticide can deem the crop dead / unsuitable for consumption among many outcomes. This data is based on crops harvested by various farmers at the end of harvest season. To simplify the problem, you can assume that all other factors like variations in farming techniques have been controlled for.

* **Agriculture Dataset Overview.**

|  |  |  |  |
| --- | --- | --- | --- |
| Column |  |  | Description |
| Id |  |  | Unique ID |
| Estimated\_Insects\_Count |  |  | Estimated insects count per square meter |
| Crop\_Type |  |  | Category of Crop(0,1) |
| Soil\_Type |  |  | Category of Soil (0,1) |
| Pesticide\_Use\_Category |  |  | Type of pesticides uses (1- Never, 2-Previously Used, 3-Currently Using) |
| Number\_Doses\_Week |  |  | Number of doses per week |
| Number\_Weeks\_Used |  |  | Number of weeks used |
| Number\_Weeks\_Quit |  |  | Number of weeks quit |
| Season |  |  | Season Category (1,2,3) |
| Crop\_Damage |  |  | Crop Damage Category (0=alive, 1=Damage due to other causes, 2=Damage due to Pesticides |

A picture containing person, outdoor

Description automatically generated

**Let’s start Analysis on Agriculture Dataset. Below you will find the way to do analysis on Agriculture dataset using Python.**

* **Importing the Libraries**

First, we need to import library to start analysis on data.

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **warnings**

warnings.filterwarnings("ignore")

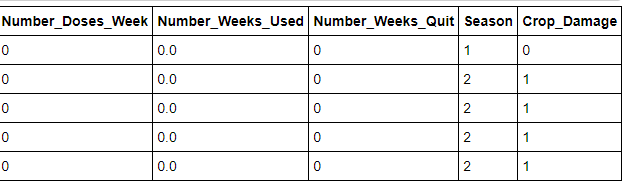
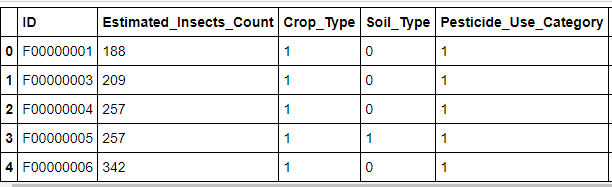
After importing libraries, we need to get data from excel and start doing analysis on it. We need to check various factor such as datatype, null values (missing values) to do more efficient analysis. If required, we also need to create additional columns for further analysis.

Now, we need to load dataset with the help of Pandas

df = pd.read\_excel("train\_agriculture.xlsx")

**Check the top 5 rows and columns of dataset**

df.head()



**Check the columns name of dataset**

df.columns

Index(['ID', 'Estimated\_Insects\_Count', 'Crop\_Type', 'Soil\_Type',

'Pesticide\_Use\_Category','Number\_Doses\_Week','Number\_Weeks\_Used',

'Number\_Weeks\_Quit', 'Season', 'Crop\_Damage']

dtype='object')

**Check the information of dataset**

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4599 entries, 0 to 4598

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 4599 non-null object

1 Estimated\_Insects\_Count 4599 non-null int64

2 Crop\_Type 4599 non-null int64

3 Soil\_Type 4599 non-null int64

4 Pesticide\_Use\_Category 4599 non-null int64

5 Number\_Doses\_Week 4599 non-null int64

6 Number\_Weeks\_Used 4157 non-null float64

7 Number\_Weeks\_Quit 4599 non-null int64

8 Season 4599 non-null int64

9 Crop\_Damage 4599 non-null int64

dtypes: float64(1), int64(8), object(1)

memory usage: 359.4+ KB

# Data Analysis and Visualisation

Let’s start with identifying null values and dropping the column ( ID ) as it will not help us in prediction.

df.drop('ID', axis=1, inplace = **True**)

df.info()

print("Now we can see there is no ID column in Data, we have dropped it")

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4599 entries, 0 to 4598

Data columns (total 9 columns):

# Column Non-Null Count Dtype

- ------ -------------- -----

0 Estimated\_Insects\_Count 4599 non-null int64

1 Crop\_Type 4599 non-null int64

2 Soil\_Type 4599 non-null int64

3 Pesticide\_Use\_Category 4599 non-null int64

4 Number\_Doses\_Week 4599 non-null int64

5 Number\_Weeks\_Used 4157 non-null float64

6 Number\_Weeks\_Quit 4599 non-null int64

7 Season 4599 non-null int64

8 Crop\_Damage 4599 non-null int64

dtypes: float64(1), int64(8)

memory usage: 323.5 KB

Now we can see there is no ID column in Data, we have dropped it

Now, we check the null values in the dataset

df.isnull().sum()

Estimated\_Insects\_Count 0

Crop\_Type 0

Soil\_Type 0

Pesticide\_Use\_Category 0

Number\_Doses\_Week 0

Number\_Weeks\_Used 442

Number\_Weeks\_Quit 0

Season 0

Crop\_Damage 0

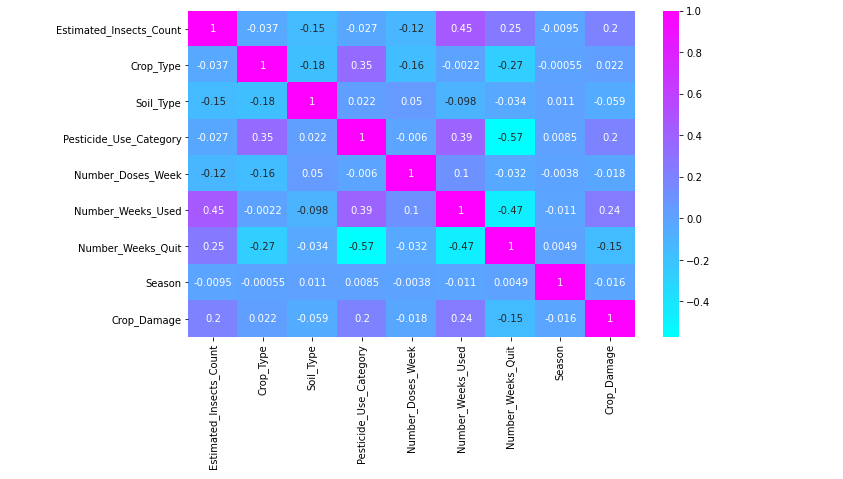
dtype: int64

**Observation: -** Only one column Number\_Weeks\_Used has missing values. Let’s find out how to handle but before which let’s see the correlation of the dataset.

plt.figure(figsize=(10,6))

sns.heatmap(df.corr(), annot = **True**, cmap = "cool")

plt.show()



**Few observations from above correlation**

Estimated\_Insects\_Count 0.45

Crop\_Type -0.0022

Soil\_Type -0.098

Pesticide\_Use\_Category 0.39

Number\_Doses\_Week 0.1

Number\_Weeks\_Quit -0.47

Season -0.011

Crop\_Damage 0.24

*We can see Number\_Weeks\_Quit -0.467 has the higest correlation with 'Number\_Weeks\_Used' we will split the data according to the unique values of Number\_Weeks\_Quit and fill null values in 'Number\_Weeks\_Used'*

**Let’s fill the null values in ‘Number\_Weeks\_Used with mode()**

**for** i **in** df.Number\_Weeks\_Quit.unique():

list.append(i)

**for** i **in** range(0,len(list)):

exec(f'NWQ\_**{**i**}** = df.loc[df.Number\_Weeks\_Quit==**{**list[i]**}**]')

exec(f"NWQ\_**{**i**}**.Number\_Weeks\_Used.fillna(NWQ\_**{**i**}**['Number\_Weeks\_Used'].mode()[0], inplace = True)")

DF = pd.concat([NWQ\_0,NWQ\_1,NWQ\_2,NWQ\_3,NWQ\_4,NWQ\_5,NWQ\_6,NWQ\_7,NWQ\_8,NWQ\_9,NWQ\_10,NWQ\_11,NWQ\_12,NWQ\_13,NWQ\_14,

NWQ\_15,NWQ\_16,NWQ\_17,NWQ\_18,NWQ\_19,NWQ\_20,NWQ\_21,NWQ\_22,NWQ\_23,NWQ\_24,NWQ\_25,NWQ\_26,NWQ\_27,

NWQ\_28,NWQ\_29,NWQ\_30,NWQ\_31,NWQ\_32,NWQ\_33,NWQ\_34,NWQ\_35,NWQ\_36,NWQ\_37,NWQ\_38,NWQ\_39,NWQ\_40,

NWQ\_41,NWQ\_42,NWQ\_43,NWQ\_44,NWQ\_45,NWQ\_46])

df= DF.sort\_index()

DF.isna().sum()

Estimated\_Insects\_Count 0

Crop\_Type 0

Soil\_Type 0

Pesticide\_Use\_Category 0

Number\_Doses\_Week 0

Number\_Weeks\_Used 0

Number\_Weeks\_Quit 0

Season 0

Crop\_Damage 0

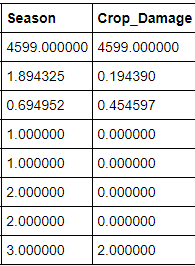
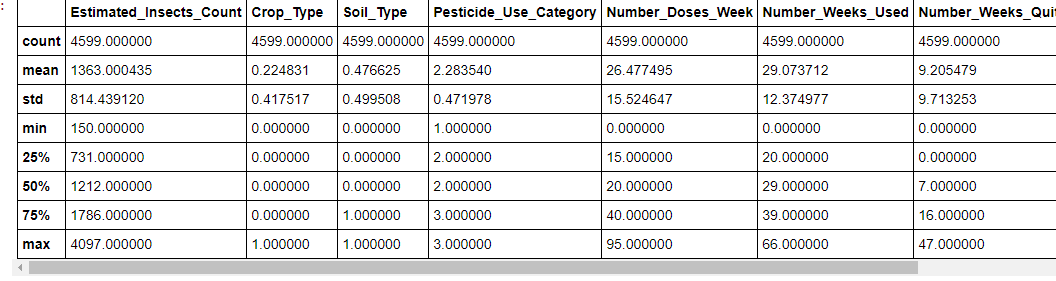
dtype: int64

**Now we can see that , there are no null values in our dataset of Agriculture.**

Now we use some Statistics in our dataset. Statistics is a collection of tools that we can use to get answers to important questions about data.

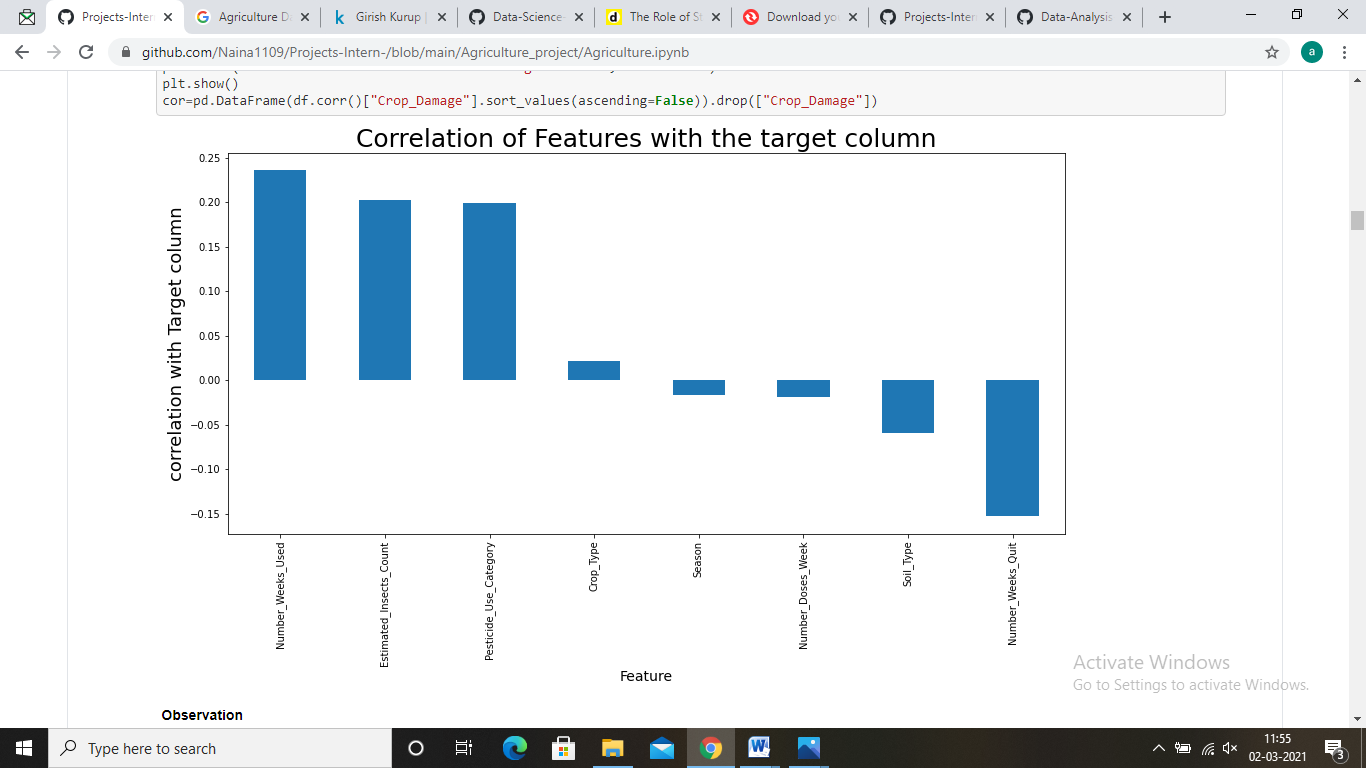
df.describe()

For this summary we are finding count , mean , std., min , 25% ,50%, 75% and Max of all Variables ) and for this summary we can understand No any null value are remains for remove , its show any outliers are present in dataset or not . What is min value, max value, standard value and Mean value of this dataset and any particular columns.

****

From above we can see that Mean is greater than 50% and also difference between 75% and Max is larger which means there are outliers and skewness present in the data.

plt.figure(figsize=(15,7)) df.corr()["Crop\_Damage"].sort\_values(ascending=**False**).drop(["Crop\_Damage"]).plot.bar() plt.xlabel("Feature", fontsize= 14) plt.ylabel("correlation with Target column", fontsize = 18) plt.title("Correlation of Features with the target column", fontsize=25) plt.show() cor=pd.DataFrame(df.corr()["Crop\_Damage"].sort\_values(ascending=**False**)).drop(["Crop\_Damage"])

****

**Observation**

* Number\_Weeks\_Used is 0.23 correlated with Crop\_Damage, which is the good number of weak pesticide used will give healthy crops at 23%
* Number\_Weeks\_Quit is -0.15 corelated with Crop\_damage, which is Number\_Weeks\_Quit without using pesticide will give Damaged crop at 15%.

plt.subplots(figsize=(6,4))

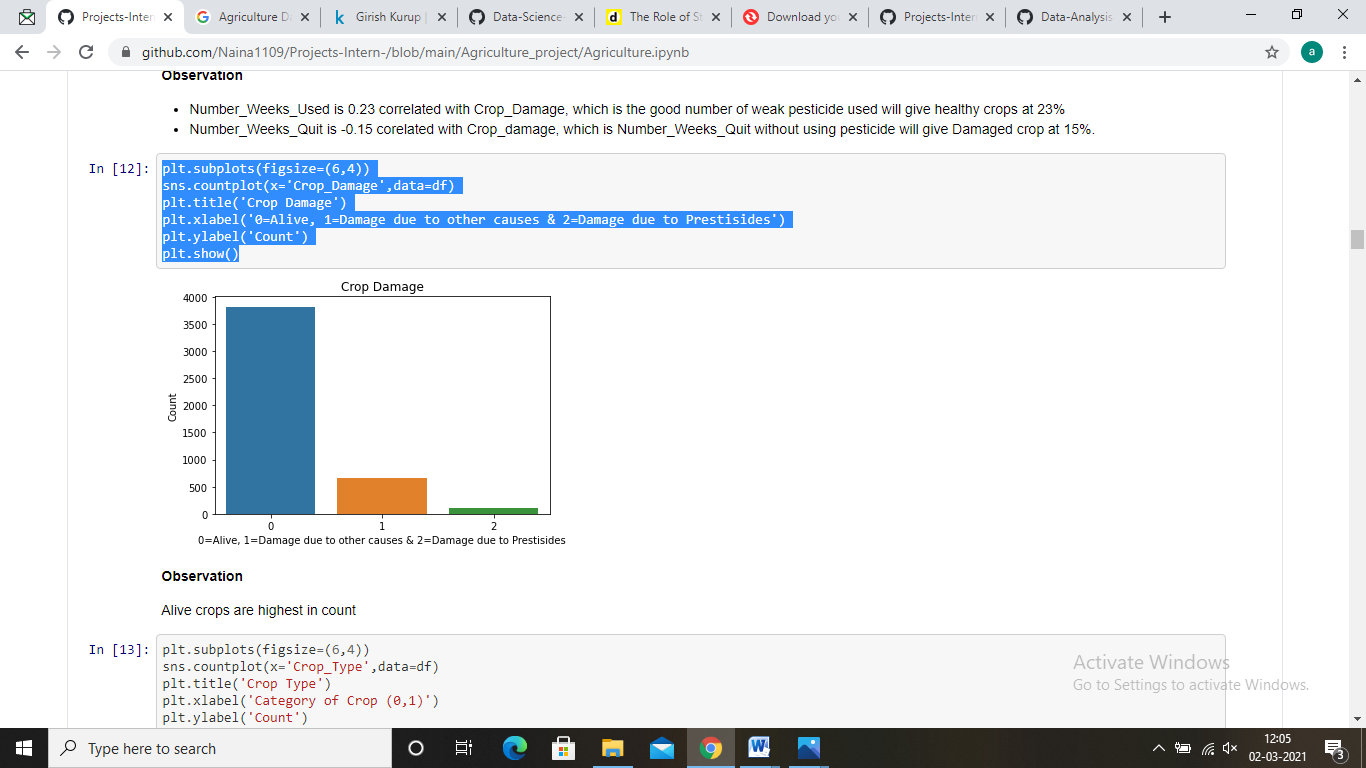
sns.countplot(x='Crop\_Damage',data=df)

plt.title('Crop Damage')

plt.xlabel('0=Alive, 1=Damage due to other causes & 2=Damage due to Prestisides')

plt.ylabel('Count')

plt.show()

****

**Observation**

Alive crops are highest in count

plt.subplots(figsize=(6,4))

sns.countplot(x='Crop\_Type',data=df)

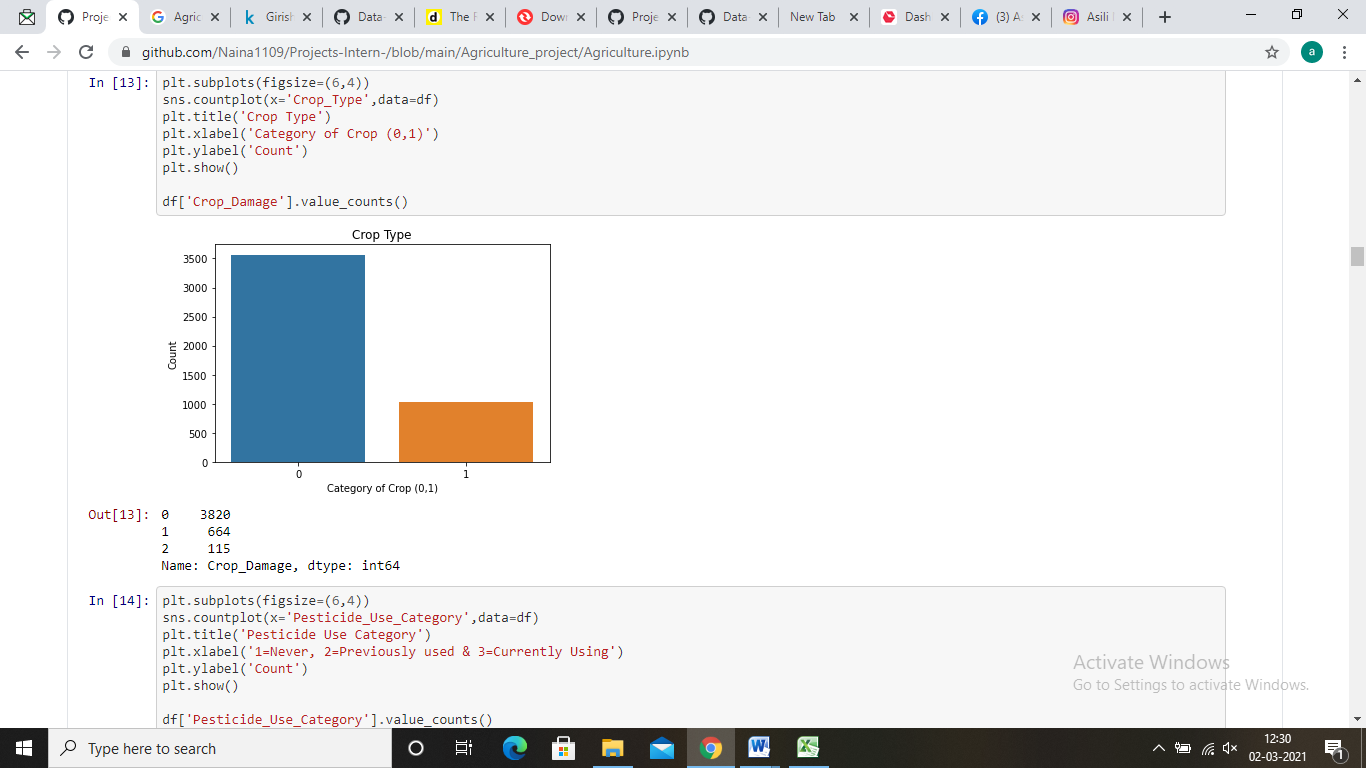
plt.title('Crop Type')

plt.xlabel('Category of Crop (0,1)')

plt.ylabel('Count')

plt.show()

df['Crop\_Damage'].value\_counts()

****

0 3820

1 664

Name: Crop\_Damage, dtype: int64

plt.subplots(figsize=(6,4))

sns.countplot(x='Pesticide\_Use\_Category',data=df)

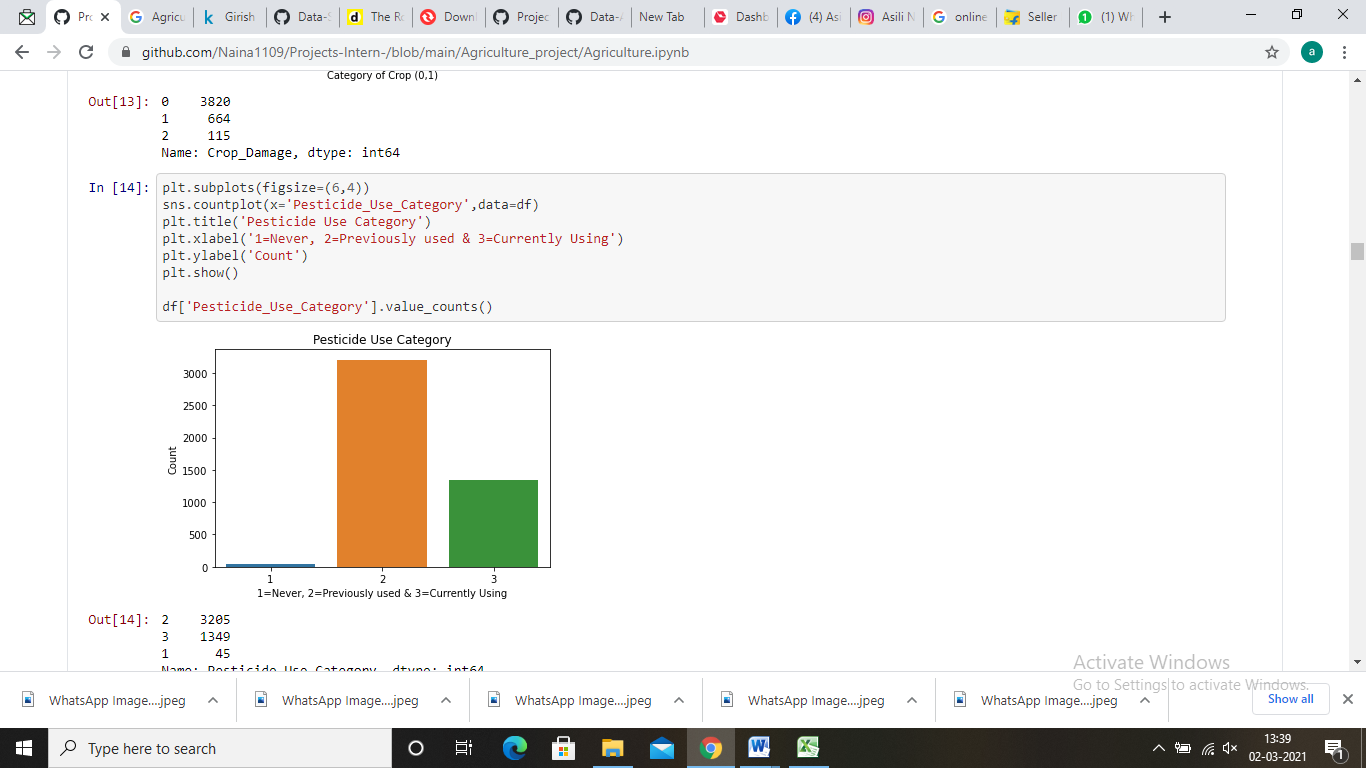
plt.title('Pesticide Use Category')

plt.xlabel('1=Never, 2=Previously used & 3=Currently Using')

plt.ylabel('Count')

plt.show()

df['Pesticide\_Use\_Category'].value\_counts()



2 3205

3 1349

1 45

Name: Pesticide\_Use\_Category, dtype: int64

plt.subplots(figsize=(6,4))

sns.countplot(x='Soil\_Type',data=df)

plt.title('Types of Soil')

plt.xlabel('Category of Soil(0,1)')

plt.ylabel('Count')

plt.show()

df['Soil\_Type'].value\_counts()

****

plt.subplots(figsize=(6,4))

sns.countplot(x='Season',data=df)

plt.title('Category of Season')

plt.xlabel('Season Category (1,2,3)')

plt.ylabel('Count')

plt.show()

df['Season'].value\_counts()

**A picture containing graphical user interface

Description automatically generated**

plt.figure(figsize=(8,4)) sns.lmplot(x='Number\_Weeks\_Used',y='Number\_Doses\_Week',fit\_reg=**False**,data=df)

plt.xlabel('Number of weeks used')

plt.title('Correlation between "Number of weeks used" and "Number of doses per week"')

plt.ylabel('Number of doses per week')

plt.show()

**Graphical user interface

Description automatically generated with medium confidence**

sns.lmplot(x='Number\_Weeks\_Used',y='Number\_Weeks\_Quit',fit\_reg=**False**,data=df)

plt.xlabel('Number of weeks used')

plt.title('Correlation between "Number of weeks used" and "Number of weeks quit"')

plt.ylabel('Number of weeks quit')

plt.show()

**Graphical user interface, text, application

Description automatically generated**

**for** col **in** df.describe().columns:

sns.distplot(df[col],color='r')

plt.show()

**A picture containing graphical user interface

Description automatically generated**

**A picture containing application

Description automatically generated**

**Histogram

Description automatically generated with medium confidence**

**Diagram

Description automatically generated with medium confidence**

**Graphical user interface, application, Word

Description automatically generated**

**Checking the Skewness in the dataset**

* **Skewness** refers to a distortion or asymmetry that deviates from the symmetrical bell curve, or normal distribution, in a set of data. ... A normal distribution has a skew of zero, while a lognormal distribution, for example, would exhibit some degree of right-skew.

df.skew()

Estimated\_Insects\_Count 0.911469

Crop\_Type 1.318693

Soil\_Type 0.093631

Pesticide\_Use\_Category 0.678586

Number\_Doses\_Week 0.945895

Number\_Weeks\_Used 0.212783

Number\_Weeks\_Quit 0.919771

Season 0.144841

Crop\_Damage 2.306933

dtype: float64

plt.figure(figsize=(20,5))

collist=["Estimated\_Insects\_Count","Number\_Weeks\_Quit","Number\_Doses\_Week","Number\_Weeks\_Used"]

**for** i **in** range (0, len(collist)):

plt.subplot(1,4,i+1)

sns.kdeplot(df[collist[i]])

plt.title(f"Skewness = **{**round(df[collist[i]].skew(),5)**}**", fontsize=15)

plt.tight\_layout()

**Graphical user interface, application

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**Skewness removal:**

Data has too much skewness but removing all will deform the data.We will use "yeojohnson method" to correct the skewness.

## **yeojohnson** estimates the optimal value of lambda for the **Yeo-Johnson** transformation. This transformation can be performed on new data, and inverted, via the predict function. The **Yeo-Johnson** is similar to the Box-Cox **method**, however it allows for the transformation of nonpositive data as well.

skewness=[]

colist = df.columns.values

**for** i **in** df.skew().values:

skewness.append(i)

df\_skewness= pd.DataFrame({"Feature\_names": colist,"Skew": skewness})

df\_skewness= df\_skewness.sort\_values(by="Skew", ascending=**False**, ignore\_index= **True**)

skew\_postive\_row= []

skew\_negative\_row=[]

**for** index, row **in** df\_skewness.iterrows():

**if** row['Skew']>0.49:

skew\_postive\_row.append(row['Feature\_names'])

**elif** row['Skew']< -0.49:

skew\_negative\_row.append(row['Feature\_names'])

print("**\n\n**Feature names with Skewness is present more than +/-0.5 as follows:**\n**","**\n\n**Postive Skewed data:**\n**",

skew\_postive\_row,"**\n\n**negative Skewed data:**\n**", skew\_negative\_row)

Feature names with Skewness is present more than +/-0.5 as follows:

Postive Skewed data:

['Crop\_Damage', 'Crop\_Type', 'Number\_Doses\_Week', 'Number\_Weeks\_Quit', 'Estimated\_Insects\_Count', 'Pesticide\_Use\_Category']

negative Skewed data:

[]

data=df

**from** **scipy.stats** **import** yeojohnson

skew = [ "Estimated\_Insects\_Count","Number\_Weeks\_Quit","Number\_Doses\_Week"]

**for** i **in** skew:

data[i]= yeojohnson(DF[i])[0]

print("BELOW GRAPH WILL SHOW THE SKEWNESS OF THE DATA")

plt.figure(figsize=(20,5))

**for** i **in** range (0, len(collist)):

plt.subplot(1,4,i+1)

plt.title(f"Skewness = **{**round(data[collist[i]].skew(),5)**}**",fontsize=20)

sns.distplot(data[collist[i]])

plt.tight\_layout()

Graphical user interface, application, Word

Description automatically generated

# Identify the outliers and remove them.

### An **outlier** is a **data** point that differs significantly from other observations. An **outlier** may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the **data** set.

To Find out the outlier in this dataset, We are going to plot the data for visualization with the help of boxplot. It is very easy way to find out the outliers in dataset

df.plot(kind='box',subplots=**True**,layout=(5,2))

Graphical user interface, text, application

Description automatically generated

**Observation**

To remove the outliers from the dataset we r using z-score threshold value.

#### A Z-score is a numerical measurement that describes a value's relationship to the mean of a group of values

**from** **scipy.stats** **import** zscore

z=np.abs(zscore(df))

z

array([[2.24476217, 1.85681868, 0.95429408, ..., 1.24022612, 1.28702725,

0.42765624],

[2.14682668, 1.85681868, 0.95429408, ..., 1.24022612, 0.15207762,

1.77233374],

[1.94644284, 1.85681868, 0.95429408, ..., 1.24022612, 0.15207762,

1.77233374],

...,

[1.00407349, 0.53855555, 1.04789501, ..., 1.69720004, 0.15207762,

0.42765624],

[2.23469653, 0.53855555, 1.04789501, ..., 1.82037081, 0.15207762,

0.42765624],

[2.34821984, 0.53855555, 1.04789501, ..., 1.82037081, 0.15207762,

0.42765624]])

threshold=3

print(np.where(z>3))

(array([ 7, 9, 14, 103, 112, 149, 165, 166, 167, 179, 186,

197, 207, 222, 223, 242, 248, 264, 265, 290, 296, 305,

316, 351, 355, 361, 374, 377, 453, 458, 466, 480, 510,

517, 522, 557, 595, 612, 634, 693, 819, 944, 980, 1067,

1178, 1327, 1339, 1340, 1737, 1807, 2043, 2139, 2170, 2171, 2237,

2489, 2585, 2590, 2663, 2671, 2696, 2710, 2716, 2723, 2767, 2792,

2796, 2797, 2804, 2806, 2814, 2828, 2829, 2834, 2841, 2842, 2851,

2860, 2861, 2870, 2884, 2888, 2912, 2915, 2917, 2921, 2923, 3015,

3045, 3058, 3086, 3107, 3126, 3183, 3189, 3206, 3248, 3628, 3629,

3657, 3686, 3716, 3717, 3718, 3740, 3796, 3850, 3851, 3935, 4091,

4155, 4277, 4278, 4378, 4545, 4570], dtype=int64), array([8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,

8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,

8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,

8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,

8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,

8, 8, 8, 8, 8, 8], dtype=int64))

df.shape

(4599, 9)

df=df[(z<3).all(axis=1)]

df.shape

(4483, 9)

**Observation:**

* 116 columns have been deleted after removing the outliers
* The percentage of data loss 2.5%
* the Loss of data is very minimal we will eliminate outliers and take the remaining data for analysis

**Model Building**

The **model building** process involves setting up ways of collecting **data**, understanding and paying attention to what is important in the **data** to answer the questions we are asking, finding a statistical, mathematical or a simulation **model** to gain understanding and make predictions.

#**splitting data into input and output variable**

x=df.drop('Crop\_Damage',axis=1) **#separating independent and target variable**

y=df['Crop\_Damage']

x.shape, y.shape

((4483, 8), (4483,))

#scaling in input variables

from sklearn.preprocessing import StandardScaler

ss=StandardScaler()

x=ss.fit\_transform(x)

**#Splitting the data into training and testing data**

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=.2,random\_state=42,stratify=y)

**Classification** is a technique where we categorize data into a given number of classes. The main goal of a **classification** problem is to identify the category/class to which a new data will fall under

**Importing Libraries for Classification of Model training and testing**.

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.svm** **import** SVC

**from** **sklearn.naive\_bayes** **import** GaussianNB

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** classification\_report,confusion\_matrix

**from** **sklearn.model\_selection** **import** GridSearchCV,cross\_val\_score

**from** **sklearn.ensemble** **import** RandomForestClassifier

**from** **sklearn.ensemble** **import** AdaBoostClassifier

**from** **sklearn.ensemble** **import** GradientBoostingClassifier

**from** **sklearn.ensemble** **import** BaggingClassifier

**from** **sklearn.ensemble** **import** ExtraTreesClassifier

model= [LogisticRegression(), GaussianNB(),SVC(), DecisionTreeClassifier(), KNeighborsClassifier(), RandomForestClassifier(), AdaBoostClassifier(), GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()]

**for** m **in** model:

m.fit(x\_train,y\_train)

m.score(x\_train,y\_train)

predm=m.predict(x\_test)

print('Accuracy score of',m,'is:')

print(accuracy\_score(y\_test,predm))

print(confusion\_matrix(y\_test,predm))

print(classification\_report(y\_test,predm))

print('===================================================================================')

print('**\n**')

Accuracy score of LogisticRegression() is:

0.8539576365663322

[[764 0]

[131 2]]

precision recall f1-score support

0 0.85 1.00 0.92 764

1 1.00 0.02 0.03 133

accuracy 0.85 897

macro avg 0.93 0.51 0.48 897

weighted avg 0.88 0.85 0.79 897

Accuracy score of GaussianNB() is:

0.8439241917502787

[[740 24]

[116 17]]

precision recall f1-score support

0 0.86 0.97 0.91 764

1 0.41 0.13 0.20 133

accuracy 0.84 897

macro avg 0.64 0.55 0.55 897

weighted avg 0.80 0.84 0.81 897

=======================================================================

Accuracy score of SVC() is:

0.855072463768116

[[763 1]

[129 4]]

precision recall f1-score support

0 0.86 1.00 0.92 764

1 0.80 0.03 0.06 133

accuracy 0.86 897

macro avg 0.83 0.51 0.49 897

weighted avg 0.85 0.86 0.79 897

=======================================================================

Accuracy score of DecisionTreeClassifier() is:

0.7859531772575251

[[666 98]

[ 94 39]]

precision recall f1-score support

0 0.88 0.87 0.87 764

1 0.28 0.29 0.29 133

accuracy 0.79 897

macro avg 0.58 0.58 0.58 897

weighted avg 0.79 0.79 0.79 897

=======================================================================

Accuracy score of KNeighborsClassifier() is:

0.835005574136009

[[725 39]

[109 24]]

precision recall f1-score support

0 0.87 0.95 0.91 764

1 0.38 0.18 0.24 133

accuracy 0.84 897

macro avg 0.63 0.56 0.58 897

weighted avg 0.80 0.84 0.81 897

=======================================================================

Accuracy score of RandomForestClassifier() is:

0.8539576365663322

[[746 18]

[113 20]]

precision recall f1-score support

0 0.87 0.98 0.92 764

1 0.53 0.15 0.23 133

accuracy 0.85 897

macro avg 0.70 0.56 0.58 897

weighted avg 0.82 0.85 0.82 897

=======================================================================

Accuracy score of AdaBoostClassifier() is:

0.8494983277591973

[[753 11]

[124 9]]

precision recall f1-score support

0 0.86 0.99 0.92 764

1 0.45 0.07 0.12 133

accuracy 0.85 897

macro avg 0.65 0.53 0.52 897

weighted avg 0.80 0.85 0.80 897

=======================================================================

Accuracy score of GradientBoostingClassifier() is:

0.8472686733556298

[[749 15]

[122 11]]

precision recall f1-score support

0 0.86 0.98 0.92 764

1 0.42 0.08 0.14 133

accuracy 0.85 897

macro avg 0.64 0.53 0.53 897

weighted avg 0.80 0.85 0.80 897

=======================================================================

Accuracy score of BaggingClassifier() is:

0.8483835005574136

[[738 26]

[110 23]]

precision recall f1-score support

0 0.87 0.97 0.92 764

1 0.47 0.17 0.25 133

accuracy 0.85 897

macro avg 0.67 0.57 0.58 897

weighted avg 0.81 0.85 0.82 897

=======================================================================

Accuracy score of ExtraTreesClassifier() is:

0.8450390189520625

[[735 29]

[110 23]]

precision recall f1-score support

0 0.87 0.96 0.91 764

1 0.44 0.17 0.25 133

accuracy 0.85 897

macro avg 0.66 0.57 0.58 897

weighted avg 0.81 0.85 0.82 897

=======================================================================

model = [LogisticRegression(), GaussianNB(),SVC(),DecisionTreeClassifier(), KNeighborsClassifier(),RandomForestClassifier() ,AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()]

**for** m **in** model:

score=cross\_val\_score(m,x,y,cv=10,scoring='accuracy')

print('model:',m)

print('score:',score)

print('Mean score',score.mean())

print('standard deviation:',score.std())

print('===================================================================================')

print('**\n**')

model: LogisticRegression()

score: [0.74832962 0.85077951 0.85077951 0.85267857 0.85267857 0.85267857

0.85267857 0.85267857 0.85267857 0.85044643]

Mean score 0.8416406498568246

standard deviation: 0.031116782577906073

===========================================================================

model: GaussianNB()

score: [0.43875278 0.82182628 0.84855234 0.85267857 0.85267857 0.84375

0.5 0.86830357 0.86607143 0.85044643]

Mean score 0.7743059974546611

standard deviation: 0.15354843844943075

===========================================================================

model: SVC()

score: [0.27171492 0.72160356 0.78173719 0.85267857 0.85267857 0.83705357

0.34821429 0.85267857 0.85267857 0.85044643]

Mean score 0.7221484250715877

standard deviation: 0.21073306485783938

===========================================================================

model: DecisionTreeClassifier()

score: [0.15144766 0.5857461 0.56792873 0.32589286 0.57589286 0.70758929

0.25892857 0.390625 0.24553571 0.85044643]

Mean score 0.46600332087177854

standard deviation: 0.21417449150055892

===========================================================================

model: KNeighborsClassifier()

score: [0.34521158 0.63919822 0.67260579 0.81696429 0.85267857 0.74776786

0.37723214 0.73660714 0.828125 0.85267857]

Mean score 0.6869069161629017

standard deviation: 0.17695392783653124

===========================================================================

model: RandomForestClassifier()

score: [0.18040089 0.65033408 0.65924276 0.78348214 0.78571429 0.77901786

0.29017857 0.59821429 0.57142857 0.85044643]

Mean score 0.614845987114222

standard deviation: 0.2096754004112163

===========================================================================

model: AdaBoostClassifier()

score: [0.65701559 0.85077951 0.85077951 0.85267857 0.85267857 0.86607143

0.58928571 0.84598214 0.75669643 0.85044643]

Mean score 0.7972413895959274

standard deviation: 0.09293867116394225

===========================================================================

model: GradientBoostingClassifier()

score: [0.15144766 0.79510022 0.76391982 0.77455357 0.78348214 0.80580357

0.29464286 0.27678571 0.27901786 0.85044643]

Mean score 0.5775199848870506

standard deviation: 0.2703761563743264

===========================================================================

model: BaggingClassifier()

score: [0.14699332 0.65701559 0.66815145 0.40178571 0.74776786 0.80133929

0.31473214 0.52008929 0.33035714 0.84375 ]

Mean score 0.5431981784918867

standard deviation: 0.22414114698043977

===========================================================================

model: ExtraTreesClassifier()

score: [0.22048998 0.6325167 0.67928731 0.734375 0.81696429 0.77232143

0.30133929 0.52455357 0.76339286 0.85044643]

Mean score 0.6295686843779829

standard deviation: 0.20536904359266786

===========================================================================

**Observation:**

* Logistic Regression and SVC is performing the best with 85% accuracy

# To Find the best Parameters, Using GridSearchCV

GridSearchCV is a function that comes in Scikit-learn’s(or SK-learn) model\_selection package. his function helps to loop through predefined hyperparameters and fit your estimator (model) on our training set. So, in the end, we can select the best parameters from the listed hyperparameters.

parameters={'C':[1,10],'random\_state':range(42,100)}

lg=LogisticRegression()

clf=GridSearchCV(lg,parameters)

clf.fit(x,y)

print(clf.best\_params\_)

{'C': 1, 'random\_state': 42}

lr=LogisticRegression(C=1,random\_state=42)

lr.fit(x\_train,y\_train)

lr.score(x\_train,y\_train)

predlr=lr.predict(x\_test)

print(accuracy\_score(y\_test,predlr))

print(confusion\_matrix(y\_test,predlr))

print(classification\_report(y\_test,predlr))

0.8539576365663322

[[764 0]

[131 2]]

precision recall f1-score support

0 0.85 1.00 0.92 764

1 1.00 0.02 0.03 133

accuracy 0.85 897

macro avg 0.93 0.51 0.48 897

weighted avg 0.88 0.85 0.79 897

score=cross\_val\_score(lr,x,y,cv=5,scoring='accuracy')

print(score)

print('Mean:',score.mean())

print('Std:',score.std())

[0.80713489 0.85172798 0.85061315 0.85267857 0.8515625 ]

Mean: 0.8427434205287465

Std: 0.017816316167771754

parameters={'kernel':('linear','rbf','poly','sigmoid'),'C':[1,10],'random\_state':range(42,60)}

sv=SVC()

clf=GridSearchCV(sv,parameters)

clf.fit(x,y)

print(clf.best\_params\_)

{'C': 1, 'kernel': 'linear', 'random\_state': 42}

sv=SVC(kernel='linear',C=1,random\_state=42,probability=True)

sv.fit(x\_train,y\_train)

sv.score(x\_train,y\_train)

predsv=sv.predict(x\_test)

print(accuracy\_score(y\_test,predsv))

print(confusion\_matrix(y\_test,predsv))

print(classification\_report(y\_test,predsv))

0.8517279821627648

[[764 0]

[133 0]]

precision recall f1-score support

0 0.85 1.00 0.92 764

1 0.00 0.00 0.00 133

accuracy 0.85 897

macro avg 0.43 0.50 0.46 897

weighted avg 0.73 0.85 0.78 897

score=cross\_val\_score(sv,x,y,cv=5,scoring='accuracy')

print(score)

print('Mean:',score.mean())

print('Std:',score.std())

[0.85172798 0.85172798 0.85172798 0.85267857 0.8515625 ]

Mean: 0.8518850035833732

Std: 0.0004019267758711051

cm=confusion\_matrix(y\_test,predlr)

sns.heatmap(cm,annot=True,cbar=False,cmap='Blues')

plt.title('confusion Matrix of GaussianNB')

plt.show()

Graphical user interface, application

Description automatically generated

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

y\_pred\_prob=lr.predict\_proba(x\_test)[:,1]

fpr,tpr,thresholds=roc\_curve(y\_test,y\_pred\_prob)

plt.plot([0,1],[0,1],'k--')

plt.plot(fpr,tpr,label='LogisticRegression')

plt.xlabel('False Positive Rate')

plt.show()

auc\_score=roc\_auc\_score(y\_test,predlr)

print(auc\_score)

Graphical user interface, application

Description automatically generated

# Conclusion:

The above study helps to understand what are the factors that affects the quality of the Crops, how to use the pesticide to get a good quality of crops. Which season will give better yield.

# Saving This Logistic Regression

import joblib

# Save the model as a pickle in a file

joblib.dump(lr,'Agriculture\_lr.pkl')

['Agriculture\_lr.pkl']