**DSML MINI PROJECT**

**MARKET DRIVEN CROP PLANNING.**

* **Problem Statement**

Develop a solution that utilizes market data and crop performance to optimize crop planning and maximize profitability for farmers

* **Introduction**

Market-driven crop planning involves leveraging market data and crop performance insights to optimize the allocation of resources and maximize profitability for farmers. By integrating real-time market trends, demand forecasts, and historical crop data, farmers can make informed decisions about which crops to cultivate, when to plant and harvest, and how much to produce. This approach empowers farmers to adapt their crop plans dynamically to meet shifting market demands, mitigate risks, and capitalize on opportunities, ultimately leading to improved efficiency, sustainability, and financial outcomes in agriculture.

* **PROPOSED SYSTEM**

**Literature Review**

|  |  |  |
| --- | --- | --- |
| **Author** | **Dataset used** | **Algorithm suggested** |
| Vamsi Tej chowdary, M. Robinson Joel ,V.Ebenezer Karunya ,Bijolin Edwin ,Roshni Thanka , Arul Jeyaraj | [All Agriculture related Datasets for India (kaggle.com)](https://www.kaggle.com/datasets/thammuio/all-agriculture-related-datasets-for-india) | Decision Tree,Naive Bayes |
| Madhuri Shripathi Rao, Arushi Singh, N.V. Subba Reddy and Dinesh U Acharya | [All Agriculture related Datasets for India (kaggle.com)](https://www.kaggle.com/datasets/thammuio/all-agriculture-related-datasets-for-india) | KNN, Decision Tree, and Random Forest |
| Mahmoud Y. Shams,Samah A. Gamel, Fatma M. Talaat | [All Agriculture related Datasets for India (kaggle.com)](https://www.kaggle.com/datasets/thammuio/all-agriculture-related-datasets-for-india) | Gradient Boosting (GB), Decision Tree (DT) ,Random Forest (RF) |
| Jyoti Deone,Rahat Afreen Siddiqui | [All Agriculture related Datasets for India (kaggle.com)](https://www.kaggle.com/datasets/thammuio/all-agriculture-related-datasets-for-india) | Random Forest  Naïve Bayes |
| Rinumoni Buragohin, P.P. Bora, J.P. Hazarika and Nivedita Deka | [All Agriculture related Datasets for India (kaggle.com)](https://www.kaggle.com/datasets/thammuio/all-agriculture-related-datasets-for-india) | SVM |

**DATA COLLECTION (Kaggle Datasets)**

* Market Data
* Crop performance Data
* Weather Data

**DATA PREPARATION**

* Integration that is integrating the collected data from various sources into a unified database.
* Cleaning the data to remove duplicates , errors that could distort analysis results.
* Feature Engineering which includes enhancement of dataset by creating new features or variables that captures relevant insights for crop planning.

**DATA PREPROCESSING**

* Handling missing values in the dataset through techniques as

imputation or deletion.

* Transforming the data using techniques such as scaling,encoding

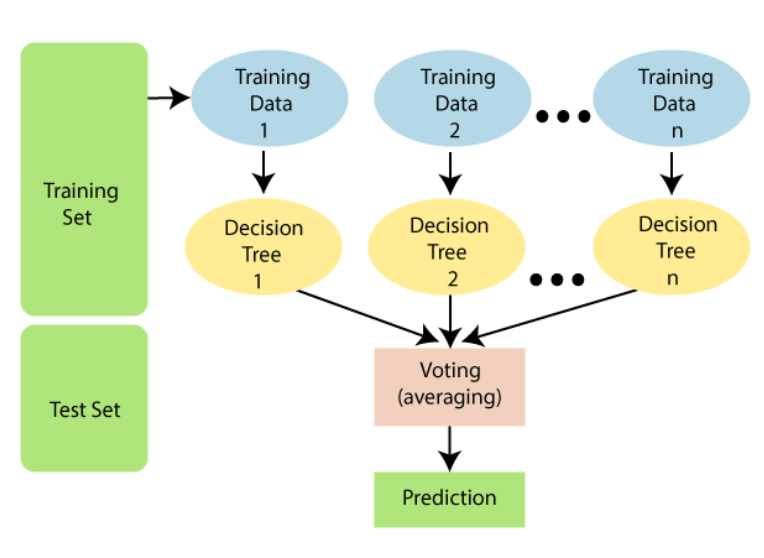
or dimensionality reduction.

* Aggregating Temporal data e.g. daily weather observations into

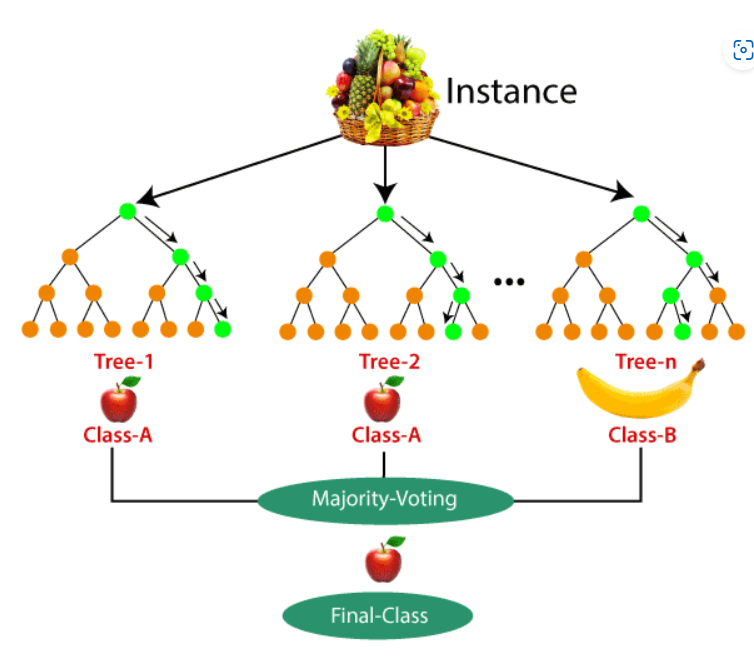
meaningful periods e.g. weekly, monthly averages.

* Validating the integrity and consistency of the preprocessed data.

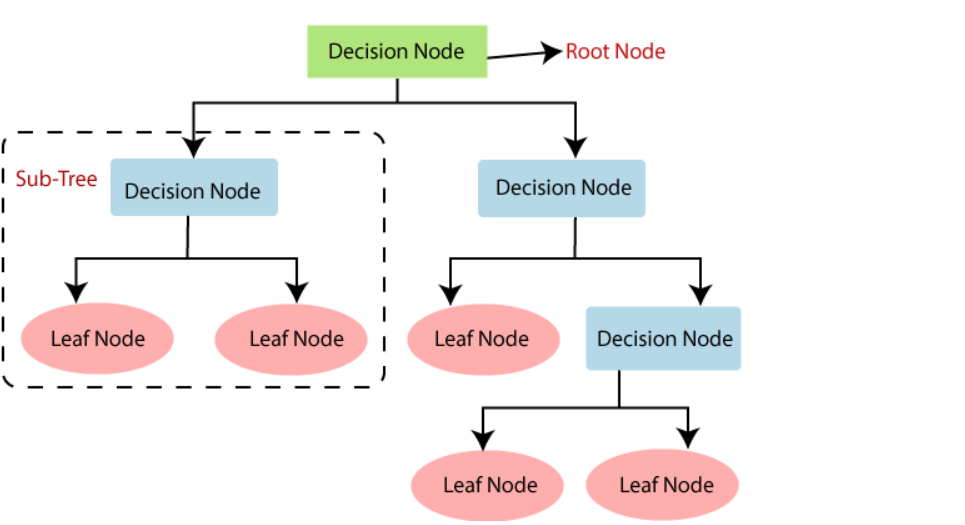
**1]RANDOM FOREST ALGORITHM**

****

* Uses multiple Decision trees for higher accuracy
* Greater the number of trees in the forest, greater the accuracy

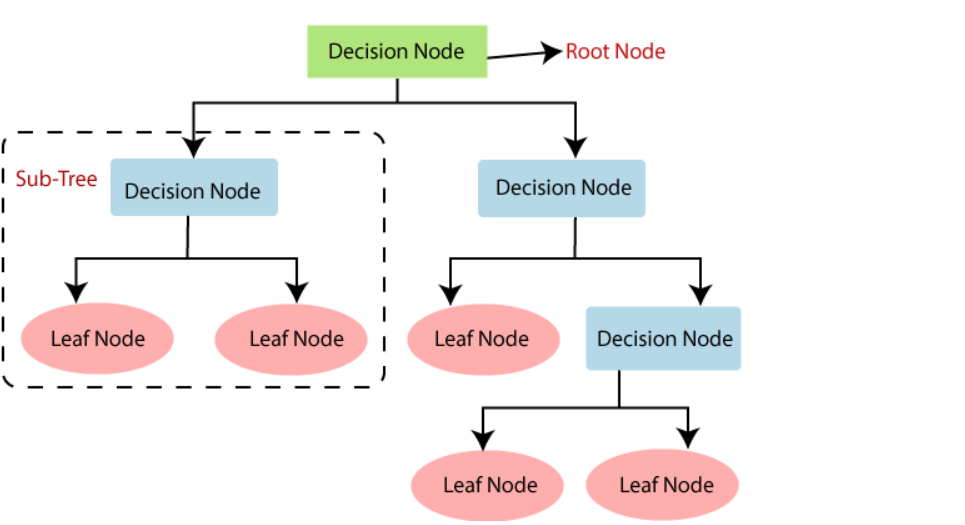


**Decision Tree Algorithm**

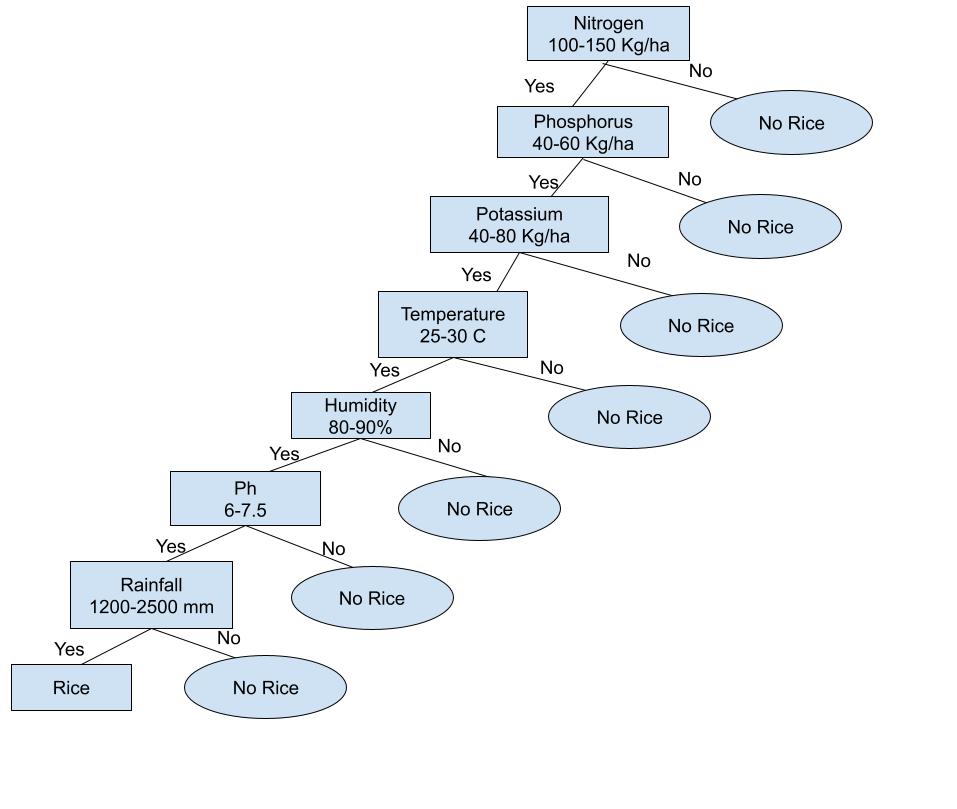
****

**2]Decision Tree Algorithm**

**Decision Tree Algorithm**

****

**Decision tree for our project**

****

**3]Naïve Bayes Classifier**

**4]Support Vector Machine Classifier**

**MODEL EVALUATION**

* We are using confusion matrix for evaluating the model accuracy
* As well testing the model on different datasets

[ ]

import pandas as pd

import numpy as np

from sklearn import datasets

[ ]

#uploading the datatset

df=pd.read\_csv("/content/Crop\_recommendation.csv")

#importing the data

print(df)

output

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

... ... .. .. ... ... ... ... ...

2195 107 34 32 26.774637 66.413269 6.780064 177.774507 coffee

2196 99 15 27 27.417112 56.636362 6.086922 127.924610 coffee

2197 118 33 30 24.131797 67.225123 6.362608 173.322839 coffee

2198 117 32 34 26.272418 52.127394 6.758793 127.175293 coffee

2199 104 18 30 23.603016 60.396475 6.779833 140.937041 coffee

Year before price(/quintal) Current price(/quintal) \

0 2944 2913

1 2959 3004

2 3140 3082

3 2992 3053

4 3082 2990

... ... ...

2195 4745 2797

2196 3402 3517

2197 2788 3415

2198 4417 4820

2199 4882 3053

Percentage increase in price recomandation

0 -1.052989 No

1 1.520784 No

2 -1.847134 No

3 2.038770 No

4 -2.985075 No

... ... ...

2195 -41.053741 No

2196 3.380364 No

2197 22.489240 coffee

2198 9.123840 coffee

2199 -37.464154 No

[2200 rows x 12 columns]

[ ]

#printing the top few rows of the dataset

df.head()

output

[ ]

#statistics about the dataset

df.describe()

output

[ ]

#removing duplicates

df=df.drop\_duplicates()

print(df)

output

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

... ... .. .. ... ... ... ... ...

2195 107 34 32 26.774637 66.413269 6.780064 177.774507 coffee

2196 99 15 27 27.417112 56.636362 6.086922 127.924610 coffee

2197 118 33 30 24.131797 67.225123 6.362608 173.322839 coffee

2198 117 32 34 26.272418 52.127394 6.758793 127.175293 coffee

2199 104 18 30 23.603016 60.396475 6.779833 140.937041 coffee

Year before price(/quintal) Current price(/quintal) \

0 2944 2913

1 2959 3004

2 3140 3082

3 2992 3053

4 3082 2990

... ... ...

2195 4745 2797

2196 3402 3517

2197 2788 3415

2198 4417 4820

2199 4882 3053

Percentage increase in price recomandation

0 -1.052989 No

1 1.520784 No

2 -1.847134 No

3 2.038770 No

4 -2.985075 No

... ... ...

2195 -41.053741 No

2196 3.380364 No

2197 22.489240 coffee

2198 9.123840 coffee

2199 -37.464154 No

[2200 rows x 12 columns]

[ ]

#statistics of data after removal of duplicates

df.describe()

output

[ ]

#finding blank values

df.isnull()

output

[ ]

#checking null values catagory wise

df.isnull().sum()

output

N 0

P 0

K 0

temperature 0

humidity 0

ph 0

rainfall 0

label 0

Year before price(/quintal) 0

Current price(/quintal) 0

Percentage increase in price 0

recomandation 0

dtype: int64

[ ]

#spliting data into independent and dependent variables

#independent variables

X=df.iloc[:,[0,1,2,3,4,5,6,8,9]].values

#dependent variable

Y= df.iloc[:, 11].values

#printing endependent variable values

print(X)

output

[[ 90. 42. 43. ... 202.9355362 2944.

2913. ]

[ 85. 58. 41. ... 226.6555374 2959.

3004. ]

[ 60. 55. 44. ... 263.9642476 3140.

3082. ]

...

[ 118. 33. 30. ... 173.3228386 2788.

3415. ]

[ 117. 32. 34. ... 127.1752928 4417.

4820. ]

[ 104. 18. 30. ... 140.9370415 4882.

3053. ]]

[ ]

#printing dependent variable values

print(Y)

output

['rice' 'rice' 'rice' ... 'coffee' 'coffee' 'coffee']

[ ]

#splitting the dataset into trainning and test set

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

#dataset is splites in 3:1 proporsion as train:test

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=0)

[ ]

#feature scaling

from sklearn.preprocessing import StandardScaler

st\_X=StandardScaler()

X\_train=st\_X.fit\_transform(X\_train)

X\_test=st\_X.transform(X\_test)

[ ]

#Random Forest algorithm

#fitting decision tree classifier to the training set

from sklearn.ensemble import RandomForestClassifier

#we generate 10 random trees to predict the output

classifier=RandomForestClassifier(n\_estimators=10,criterion="entropy")

classifier.fit(X\_train,Y\_train)

output

[ ]

#predicting the test set results

Y\_pred=classifier.predict(X\_test)

print(Y\_pred)

output

['No' 'watermelon' 'No' 'No' 'blackgram' 'No' 'mothbeans' 'No' 'No' 'No'

'mothbeans' 'No' 'No' 'No' 'mango' 'apple' 'coffee' 'lentil' 'coffee'

'No' 'No' 'blackgram' 'kidneybeans' 'No' 'cotton' 'No' 'lentil' 'orange'

'mothbeans' 'kidneybeans' 'No' 'No' 'No' 'No' 'No' 'cotton' 'mango' 'No'

'watermelon' 'mothbeans' 'No' 'blackgram' 'No' 'watermelon' 'No' 'No'

'No' 'kidneybeans' 'No' 'No' 'mothbeans' 'No' 'No' 'No' 'No' 'No' 'No'

'No' 'lentil' 'cotton' 'No' 'No' 'watermelon' 'No' 'muskmelon' 'No' 'No'

'coffee' 'apple' 'orange' 'coconut' 'pomegranate' 'kidneybeans' 'No' 'No'

'mothbeans' 'chickpea' 'No' 'jute' 'mango' 'No' 'No' 'watermelon'

'watermelon' 'muskmelon' 'cotton' 'No' 'lentil' 'No' 'papaya' 'blackgram'

'jute' 'No' 'grapes' 'No' 'No' 'coffee' 'jute' 'No' 'No' 'orange' 'jute'

'No' 'No' 'No' 'rice' 'watermelon' 'No' 'pigeonpeas' 'No' 'muskmelon'

'No' 'coffee' 'No' 'orange' 'No' 'No' 'No' 'No' 'jute' 'No' 'No' 'orange'

'chickpea' 'No' 'blackgram' 'No' 'orange' 'No' 'No' 'mothbeans'

'mothbeans' 'muskmelon' 'No' 'No' 'mungbean' 'coconut' 'pomegranate' 'No'

'blackgram' 'No' 'No' 'apple' 'coffee' 'No' 'apple' 'jute' 'No' 'No' 'No'

'orange' 'coconut' 'mothbeans' 'No' 'pomegranate' 'No' 'watermelon'

'blackgram' 'apple' 'jute' 'lentil' 'No' 'jute' 'kidneybeans'

'kidneybeans' 'muskmelon' 'No' 'muskmelon' 'No' 'orange' 'No' 'apple'

'No' 'coconut' 'cotton' 'No' 'No' 'No' 'No' 'orange' 'No' 'papaya'

'papaya' 'No' 'kidneybeans' 'banana' 'No' 'No' 'No' 'papaya' 'No' 'jute'

'mothbeans' 'No' 'No' 'coffee' 'No' 'No' 'No' 'No' 'coconut' 'No' 'No'

'mothbeans' 'mothbeans' 'orange' 'muskmelon' 'No' 'No' 'banana' 'No' 'No'

'pigeonpeas' 'orange' 'mothbeans' 'No' 'mango' 'papaya' 'No'

'pomegranate' 'rice' 'No' 'blackgram' 'No' 'blackgram' 'No' 'No' 'No'

'orange' 'No' 'mothbeans' 'mungbean' 'orange' 'kidneybeans' 'No'

'coconut' 'mango' 'No' 'cotton' 'No' 'chickpea' 'No' 'apple' 'No' 'No'

'rice' 'watermelon' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'jute' 'papaya'

'pomegranate' 'No' 'mango' 'No' 'jute' 'watermelon' 'pomegranate' 'No'

'No' 'No' 'jute' 'No' 'chickpea' 'No' 'No' 'papaya' 'No' 'No' 'No' 'No'

'No' 'orange' 'pomegranate' 'No' 'No' 'No' 'pomegranate' 'muskmelon'

'mothbeans' 'mango' 'lentil' 'banana' 'No' 'mango' 'No' 'No' 'cotton'

'No' 'papaya' 'No' 'grapes' 'pigeonpeas' 'No' 'jute' 'No' 'chickpea' 'No'

'coffee' 'watermelon' 'No' 'jute' 'No' 'No' 'No' 'blackgram' 'No'

'mothbeans' 'No' 'blackgram' 'mango' 'No' 'mango' 'pomegranate' 'jute'

'orange' 'muskmelon' 'No' 'No' 'cotton' 'papaya' 'No' 'kidneybeans'

'lentil' 'apple' 'No' 'muskmelon' 'apple' 'papaya' 'blackgram' 'jute'

'No' 'mothbeans' 'No' 'grapes' 'jute' 'No' 'No' 'No' 'No' 'No' 'No'

'coffee' 'muskmelon' 'mothbeans' 'No' 'papaya' 'No' 'No' 'No'

'pigeonpeas' 'No' 'No' 'watermelon' 'No' 'No' 'mothbeans' 'No' 'No'

'coconut' 'No' 'kidneybeans' 'No' 'cotton' 'No' 'pomegranate' 'cotton'

'cotton' 'apple' 'cotton' 'apple' 'orange' 'No' 'No' 'banana' 'No'

'mothbeans' 'No' 'No' 'No' 'rice' 'lentil' 'No' 'No' 'No' 'No'

'muskmelon' 'No' 'apple' 'lentil' 'No' 'No' 'blackgram' 'muskmelon'

'mothbeans' 'mango' 'No' 'pomegranate' 'mango' 'chickpea' 'coconut' 'No'

'No' 'pigeonpeas' 'papaya' 'jute' 'lentil' 'cotton' 'No' 'banana'

'coconut' 'No' 'blackgram' 'No' 'orange' 'watermelon' 'No' 'apple' 'No'

'pigeonpeas' 'No' 'chickpea' 'mango' 'No' 'pomegranate' 'No' 'mango' 'No'

'No' 'No' 'No' 'No' 'cotton' 'blackgram' 'apple' 'apple' 'jute' 'jute'

'No' 'No' 'No' 'No' 'banana' 'kidneybeans' 'No' 'No' 'No' 'No' 'jute'

'No' 'No' 'mungbean' 'blackgram' 'jute' 'orange' 'pigeonpeas' 'No' 'No'

'mothbeans' 'watermelon' 'No' 'watermelon' 'papaya' 'No' 'mungbean' 'No'

'pomegranate' 'orange' 'mothbeans' 'apple' 'No' 'coconut' 'No' 'coconut'

'grapes' 'No' 'No' 'chickpea' 'No' 'No' 'No' 'No' 'mungbean' 'papaya'

'pigeonpeas' 'blackgram' 'papaya' 'muskmelon' 'blackgram' 'No' 'rice'

'coffee' 'cotton' 'grapes' 'No' 'No' 'No' 'No' 'grapes' 'watermelon' 'No'

'No' 'jute' 'No' 'cotton' 'watermelon' 'pomegranate' 'coffee' 'coconut'

'No' 'No' 'mungbean' 'No' 'watermelon' 'No' 'apple' 'apple' 'watermelon'

'No' 'mothbeans' 'mungbean' 'apple' 'mungbean' 'cotton' 'No' 'papaya'

'cotton' 'papaya' 'chickpea' 'apple' 'pomegranate' 'mothbeans' 'No'

'blackgram' 'mango' 'orange' 'jute' 'banana' 'papaya' 'coffee' 'cotton'

'mango' 'No' 'No' 'pomegranate']

[ ]

#creating the confusion matrix

from sklearn.metrics import confusion\_matrix,accuracy\_score

cm=confusion\_matrix(Y\_test,Y\_pred)

print(cm)

#accuracy of the model

accuracy\_score(Y\_test, Y\_pred)

output

[[200 1 2 1 1 0 2 3 1 0 1 0 0 1 2 0 1 2

0 0 2 0 4]

[ 0 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 5 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 3 0 0 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 4 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 16 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 24 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 15 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 0 0 0 0 0 0 0 0 0 23 0 0 0

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0

0 0 0 0 0]

[ 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18

0 0 0 0 0]

[ 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

18 0 0 0 0]

[ 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 8 0 0 0]

[ 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 14 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 5 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 15]]

0.8618181818181818

#Decision Tree algorithm

#fitting decision tree classifier to the training set

from sklearn.tree import DecisionTreeClassifier

classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0)

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



output

[24]

0s

#Decision Tree algorithm

#predicting the test set results

y\_pred=classifier.predict(x\_test)

# print(y\_pred)

#creating the confusion matrix

from sklearn.metrics import confusion\_matrix

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

print(cm)

print()

#accuracy of the model

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred)



output

[[206 1 2 2 2 0 1 0 1 0 1 2 0 0 1 2 0 1

1 0 0 0 1]

[ 1 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 5 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 18 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 4 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 15 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 24 0 0 0 0 0 0 0 2

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 16 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 1 0 0 0 0 0 0 0 23 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 14 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

20 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 13 0 0 0]

[ 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

0 0 11 0 0]

[ 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 4 0]

[ 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 13]]

0.8781818181818182

[25]

0s

#Naive Bayes Algorithm  
  
#fitting naive bayes algorithm to the training set  
from sklearn.naive\_bayes import GaussianNB    
classifier = GaussianNB()    
classifier.fit(x\_train, y\_train)

output

[27]

0s

#Naive Bayes Algorithm  
  
#predicting the test set results  
y\_pred = classifier.predict(x\_test)   
# print(y\_pred)  
  
#creating the confusion matrix  
from sklearn.metrics import confusion\_matrix    
cm = confusion\_matrix(y\_test, y\_pred)    
print(cm)  
print()  
  
#accuracy of the model  
from sklearn.metrics import accuracy\_score  
accuracy\_score(y\_test, y\_pred)

output

[[39 5 14 5 16 5 13 5 16 1 12 6 2 8 4 16 10 8 5 4 10 12 8]

[ 0 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 27 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 16 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 17 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 23 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 14 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16]]

0.6618181818181819

[28]

1s

# Support Vector Machine classifier  
  
# Support vector classifier   
from sklearn.svm import SVC   
  
#fitting support vector classifier to the training set  
classifier = SVC(kernel='linear', random\_state=0)    
classifier.fit(x\_train, y\_train)

output

[29]

# Support Vector Machine classifier

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

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

print(cm)

print()

#accuracy of the model

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred)



output

[[190 0 0 0 0 5 1 4 0 0 0 0 0 0 0 0 10 6

4 0 0 0 4]

[ 0 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 16 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 27 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 2 0 0 0 0 0 0 0 7 0 0 0 0 0 0

0 0 0 0 0]

[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 16 0 0 0 0

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 1 0 0 21 0 0 0

0 0 0 0 0]

[ 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 17 0

0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19

0 0 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

20 0 0 0 0]

[ 12 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 0 0 0]

[ 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 15 0 0]

[ 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0]

[ 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 14]]

0.7327272727272728