CROP SUGGESTION ANALYSIS

Precision agriculture is a modern farming technique that uses the data of soil characteristics, soil types, crop yield data, weather conditions and suggests the farmers with the most optimal crop to grow in their farms for maximum yield and profit.

This technique can reduce the crop failures and will help the farmers to take informed decision about their farming strategy.

DATA FIELDS MENTIONED

- P - ratio of Phosphorous content in soil
- N - ratio of Nitrogen content in soil
- K - ratio of Potassium content in soil
- humidity - relative humidity in %
- ph - ph value of the soil
- rainfall - rainfall in mm
- temperature - temperature in degree Celsius

Importing necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn import tree
```

```
import warnings
warnings.filterwarnings('ignore')
```

DATASET UPLOADING

```
df=pd.read_csv("/content/Crop_recommendation.csv")
df
```

	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

2200 rows × 8 columns

```
df.dtypes
```

```
N                int64
P                int64
K                int64
temperature    float64
humidity       float64
ph             float64
rainfall       float64
label           object
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

```
df.describe()
```

```
df.isna().sum()
```

```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

75% 84.250000 68.000000 49.000000 28.561654 89.948771 6.923643 124

```
df.size
```

```
17600
```

```
df.shape
```

```
(2200, 8)
```

```
df.columns
```

```
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label'], dtype='object')
```

```
df["label"].unique()
print(len(df["label"].unique()))
```

```
22
```

```
df["label"].value_counts()
```

```
rice           100
maize          100
jute           100
cotton         100
coconut        100
papaya         100
orange         100
apple          100
muskmelon      100
watermelon     100
grapes         100
mango          100
banana         100
pomegranate    100
lentil         100
blackgram      100
mungbean       100
mothbeans      100
pigeonpeas     100
kidneybeans    100
chickpea       100
coffee         100
Name: label, dtype: int64
```

PIVOT TABLE The pivot table function takes in a data frame and the parameters detailing the shape for the necessary data.Then it outputs summarized data in the form of a pivot table

```
cs=pd.pivot_table(df,index=['label'],aggfunc='mean')
cs_new=cs.reset_index()
cs_new
```

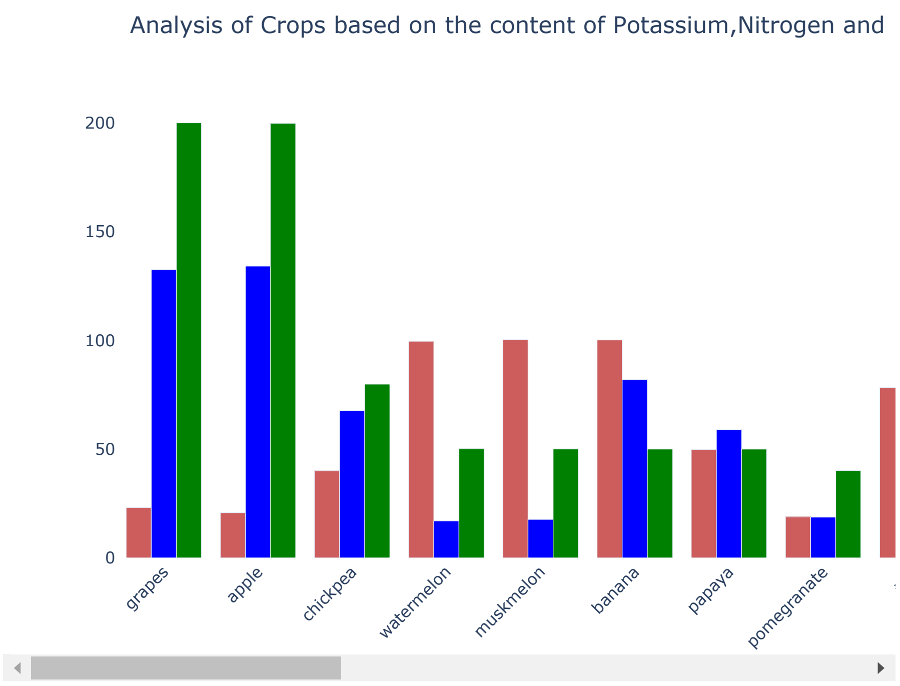
	label	K	N	P	humidity	ph	rainfall	temperature
0	apple	199.89	20.80	134.22	92.333383	5.929663	112.654779	22.630942
1	banana	50.05	100.23	82.01	80.358123	5.983893	104.626980	27.376798
2	blackgram	19.24	40.02	67.47	65.118426	7.133952	67.884151	29.973340
3	chickpea	79.92	40.09	67.79	16.860439	7.336957	80.058977	18.872847
4	coconut	30.59	21.98	16.93	94.844272	5.976562	175.686646	27.409892
5	coffee	29.94	101.20	28.74	58.869846	6.790308	158.066295	25.540477
6	cotton	19.56	117.77	46.24	79.843474	6.912675	80.398043	23.988958
7	grapes	200.11	23.18	132.53	81.875228	6.025937	69.611829	23.849575
8	jute	39.99	78.40	46.86	79.639864	6.732778	174.792798	24.958376
9	kidneybeans	20.05	20.75	67.54	21.605357	5.749411	105.919778	20.115085
10	lentil	19.41	18.77	68.36	64.804785	6.927932	45.680454	24.509052
11	maize	19.79	77.76	48.44	65.092249	6.245190	84.766988	22.389204
12	mango	29.92	20.07	27.18	50.156573	5.766373	94.704515	31.208770
13	mothbeans	20.23	21.44	48.01	53.160418	6.831174	51.198487	28.194920
14	mungbean	19.87	20.99	47.28	85.499975	6.723957	48.403601	28.525775
15	muskmelon	50.08	100.32	17.72	92.342802	6.358805	24.689952	28.663066
16	orange	10.01	19.58	16.55	92.170209	7.016957	110.474969	22.765725
17	papaya	50.04	49.88	59.05	92.403388	6.741442	142.627839	33.723859
18	pigeonpeas	20.29	20.73	67.73	48.061633	5.794175	149.457564	27.741762
19	pomegranate	40.21	18.87	18.75	90.125504	6.429172	107.528442	21.837842
20	rice	39.87	79.89	47.58	82.272822	6.425471	236.181114	23.689332
21	watermelon	50.22	99.42	17.00	85.160375	6.495778	50.786219	25.591767

DATA VISUALIZATION AND ANALYSIS

Evaluation of crops based on the values of N,K,P

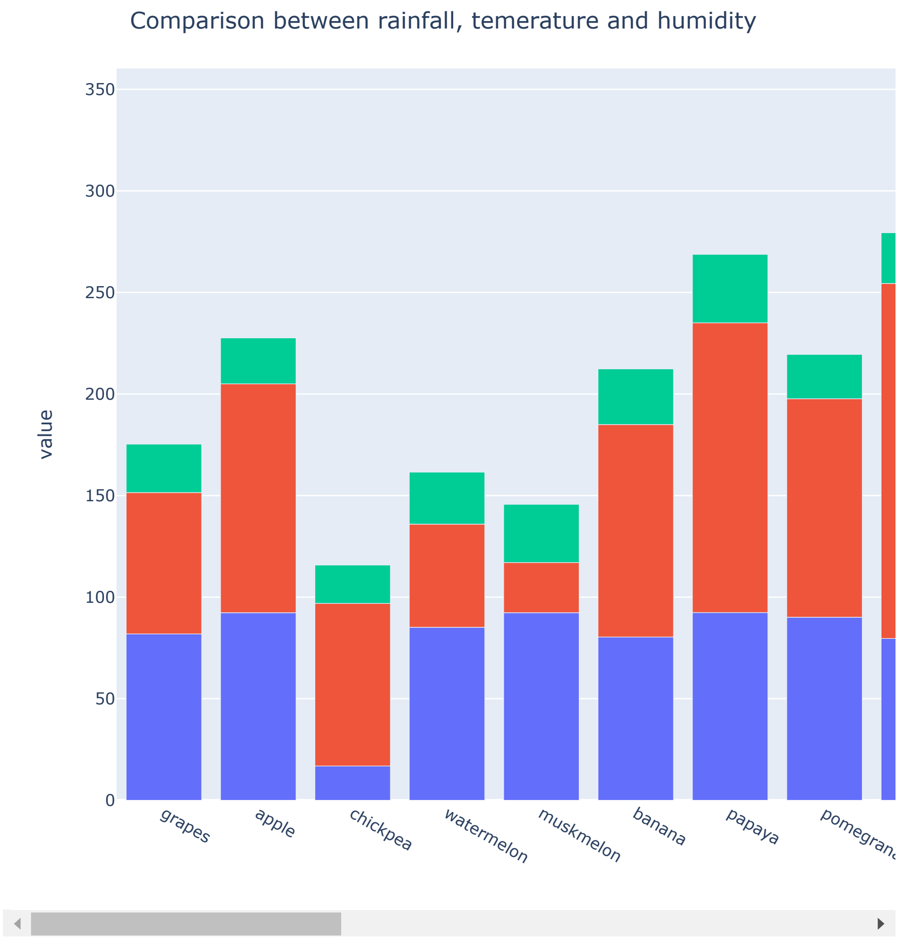
```
import plotly.graph_objects as go

cs=cs.sort_values(by='N', ascending=False)
```

```
cs=cs.sort_values(by='P', ascending=False)
cs=cs.sort_values(by='K', ascending=False)

fig=go.Figure()
fig.add_trace(go.Bar(x=cs.index,y=cs['N'],name='Nitrogen',marker_color='indianred'))
fig.add_trace(go.Bar(x=cs.index,y=cs['P'],name='Phosphorous',marker_color='blue'))
fig.add_trace(go.Bar(x=cs.index,y=cs['K'],name='Potassium',marker_color='green'))

fig.update_layout(title="Analysis of Crops based on the content of Potassium,Nitrogen and Phosporous",plot_bgcolor='white',barmode='group',xa
fig.show()
```



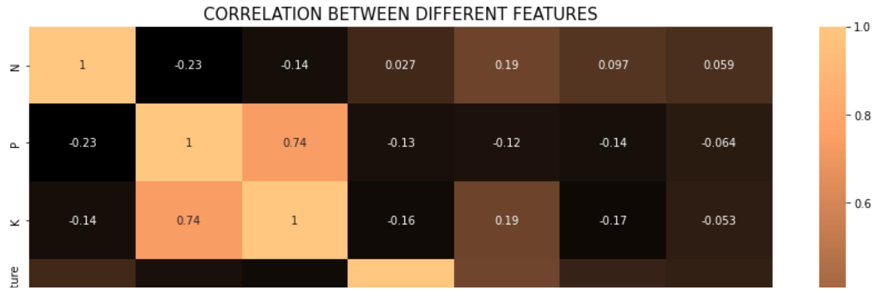
Evaluation of crops based on natural calamities

```
import plotly.express as px

fig=px.bar(cs,x=cs.index,y=["humidity","rainfall","temperature"])
fig.update_layout(title_text="Comparison between rainfall, temerature and humidity",height=700)
fig.show()
```



HEATMAP DISPLAYING THE CORRELATION BETWEEN DIFFERENT FEATURES

```
fig,ax=plt.subplots(1,1,figsize=(15,9))
sns.heatmap(df.corr(),annot=True,cmap="copper")
plt.title('CORRELATION BETWEEN DIFFERENT FEATURES',fontsize = 15,c='black')
plt.show()
```



DECLARING INDEPENDENT AND TARGET VARIABLES

```
features=df[['N','P','K','temperature','humidity','ph','rainfall']]
target=df['label']
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['label']=le.fit_transform(df['label'])
df
```

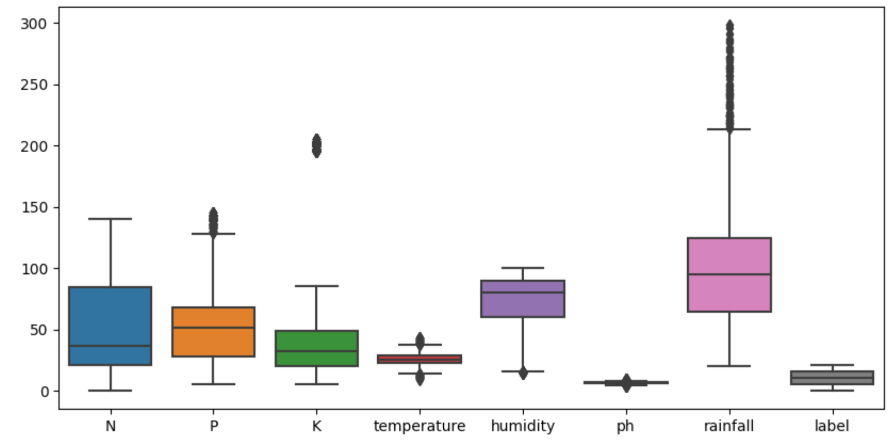
	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	20
1	85	58	41	21.770462	80.319644	7.038096	226.655537	20
2	60	55	44	23.004459	82.320763	7.840207	263.964248	20
3	74	35	40	26.491096	80.158363	6.980401	242.864034	20
4	78	42	42	20.130175	81.604873	7.628473	262.717340	20
...	...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	5
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	5
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	5
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	5
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	5

2200 rows × 8 columns

OUTLIER DETECTION AND HANDLING

```
plt.figure(figsize=[10,5],dpi=100)
sns.boxplot(data=df)
```

<AxesSubplot:>




```
for i in df["P"]:
  q1=df["P"].quantile(0.25)
  q2=df["P"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail:
    df["P"]=df["P"].replace(i,np.mean(df["P"]))
```

```
for i in df["K"]:
  q1=df["K"].quantile(0.25)
  q2=df["K"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail:
    df["K"]=df["K"].replace(i,np.mean(df["K"]))
```

```
for i in df["temperature"]:
  q1=df["temperature"].quantile(0.25)
  q2=df["temperature"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail or i < lower_tail:
    df["K"]=df["temperature"].replace(i,0)
```

```
for i in df["humidity"]:
  q1=df["humidity"].quantile(0.25)
  q2=df["humidity"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail:
    df["humidity"]=df["humidity"].replace(i,np.mean(df["humidity"]))
```

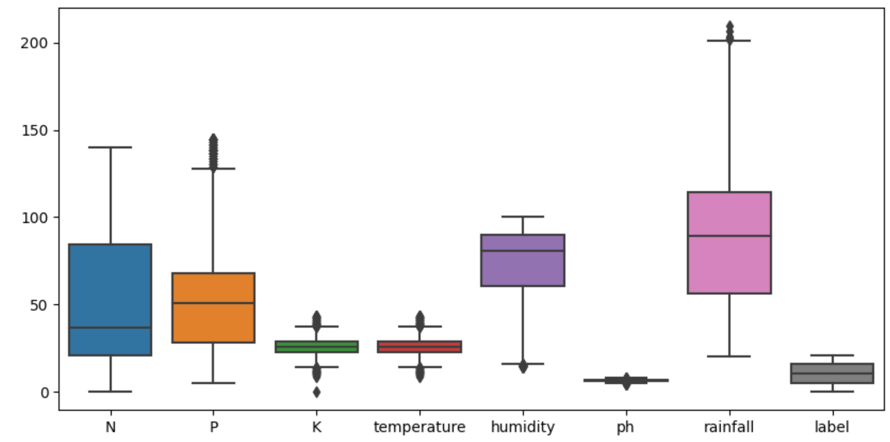
```
for i in df["ph"]:
  q1=df["ph"].quantile(0.25)
  q2=df["ph"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail or i < lower_tail:
    df["ph"]=df["ph"].replace(i,np.mean(df["ph"]))
```

```
for i in df["rainfall"]:
  q1=df["rainfall"].quantile(0.25)
  q2=df["rainfall"].quantile(0.75)
  iqr=q2-q1
  lower_tail=q1-1.5*iqr
  upper_tail=q2+1.5*iqr
  if i > upper_tail:
    df["rainfall"]=df["rainfall"].replace(i,np.median(df["K"]))
```

BOX PLOT REPRESENTATION AFTER HANDLING OUTLIERS

```
plt.figure(figsize=[10,5],dpi=100)
sns.boxplot(data=df)
```

```
<AxesSubplot:>
```



SPLITTING TRAIN AND TEST DATASET

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(features,target,test_size=0.2,random_state=2)
```

```
X_train.shape
```

```
(1760, 7)
```

```
X_train
```

	N	P	K	temperature	humidity	ph	rainfall
1936	113	38	25	22.000851	79.472710	7.388266	90.422242
610	28	35	22	29.530376	86.733460	7.156563	59.872321
372	11	61	21	18.623288	23.024103	5.532101	135.337803
1559	29	139	205	23.641424	93.744615	6.155939	116.691218
1500	24	128	196	22.750888	90.694892	5.521467	110.431786
...	...	...	...	...	...	...	...
1071	105	88	54	25.787498	84.511942	6.020445	114.200546
433	27	71	23	23.453790	46.487148	7.109598	150.871220
674	23	39	22	29.256493	81.979522	6.864839	42.024833
1099	117	81	53	29.507046	78.205856	5.507642	98.125658
1608	39	24	14	30.554726	90.903438	7.189260	106.071198

1760 rows × 7 columns

```
X_test.shape
```

```
(440, 7)
```

```
X_test
```

	N	P	K	temperature	humidity	ph	rainfall
2121	83	21	28	25.567483	60.492446	7.466901	190.225784
960	1	27	36	23.985988	93.342366	5.684995	104.991282
952	23	5	44	21.207254	94.263047	7.163005	107.566080
1958	116	52	19	22.942767	75.371706	6.114526	67.080226
681	6	37	17	28.086572	80.350059	6.760694	38.144768
...	...	...	...	...	...	...	...
1684	7	17	10	10.164313	91.223210	6.465913	106.362551
1477	86	18	45	28.965866	90.718329	6.566759	22.258381

NORMALIZING THE DATASET

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X_train)
X_train_new=sc.transform(X_train)
X_test_new=sc.transform(X_test)
```

```
X_train_new
```

```
array([[ 1.69991833, -0.48450066, -0.46654299, ...,  0.36534732,
         1.20356233, -0.23551865],
       [-0.60720953, -0.57443434, -0.52452593, ...,  0.69004063,
         0.90389393, -0.79201637],
       [-1.0686351 ,  0.20499088, -0.54385357, ..., -2.15897647,
        -1.19707381,  0.58266376],
       ...,
       [-0.74292293, -0.45452277, -0.52452593, ...,  0.4774494 ,
         0.52659831, -1.11712642],
       [ 1.80848906,  0.80454873,  0.07463107, ...,  0.30869492,
        -1.22870711, -0.09519312],
       [-0.30864004, -0.90419116, -0.67914709, ...,  0.87651775,
         0.94618139,  0.04954294]])
```

```
X_test_new
```

```
array([[ 0.88563791, -0.99412484, -0.40856006, ..., -0.48343061,
         1.30526315,  1.58250387],
       [-1.34006191, -0.81425748, -0.2539389 , ...,  0.98558411,
        -0.99933038,  0.02987117],
       [-0.74292293, -1.47377113, -0.09931774, ...,  1.02675604,
         0.91222502,  0.07677373],
       ...,
       [-1.2043485 ,  0.29492455, -0.50519828, ..., -0.17432586,
         0.78581408, -1.22347233],
       [-0.3629254 ,  0.05510141, -0.46654299, ..., -2.31192924,
        -0.88324111,  0.42580083],
       [ 1.048494  , -0.03483227, -0.09931774, ...,  0.02746888,
         1.11492982,  0.8543021 ]])
```

LIST OF ALGORITHMS IMPLEMENTED

- SUPPORT VECTOR MACHINE
- DECISION TREE
- RANDOM FOREST
- LIGHTGBM
- K-Nearest Neighbors
- NAIVE BAYES CLASSIFIER
- LOGISTIC REGRESSION

Initializing empty lists to append all modelsand corresponding name

```
acc=[]
model=[]
```

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
SVM=SVC(kernel="linear")
SVM.fit(X_train,Y_train)
Y_Pred=SVM.predict(X_test)

a=metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("SVM")
print("Accuracy Score accquired from SVM:",a*100)
```

```
Accuracy Score accquired from SVM: 97.72727272727273
```

```
from sklearn.model_selection import cross_val_score
score=cross_val_score(SVM,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9854545454545454
```

DECISION TREE ALGORITHM

```
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier(criterion="entropy")
DT.fit(X_train,Y_train)
Y_Pred=DT.predict(X_test)

a = metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("DT")
print("Accuracy Score accquired from Decision Tree:",a*100)
```

```
print("Accuracy Score accquired from Decision Tree:",a*100)
```

```
Accuracy Score accquired from Decision Tree: 98.4090909090909
```

```
score=cross_val_score(DT,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9868181818181817
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier(n_estimators=150)
RF.fit(X_train,Y_train)
Y_Pred=RF.predict(X_test)

a=metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("RF")
print("Accuracy Score accquired from Random Forest:",a*100)
```

```
Accuracy Score accquired from Random Forest: 99.54545454545455
```

```
score=cross_val_score(RF,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9936363636363638
```

LightGBM Model

```
import lightgbm as lgb
LGB= lgb.LGBMClassifier()
LGB.fit(X_train,Y_train)
Y_Pred=LGB.predict(X_test)

a = metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("LGB")
print("Accuracy acquired from LightGBM:",a*100)
```

```
Accuracy acquired from LightGBM: 99.0909090909091
```

```
score=cross_val_score(LGB,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9904545454545456
```

K-Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
Knn=KNeighborsClassifier(n_neighbors=3)
Knn.fit(X_train,Y_train)
Y_Pred=Knn.predict(X_test)

a=metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("Knn")
print("Accuracy acquired from K-Nearest Neighbors:",a*100)
```

```
Accuracy acquired from K-Nearest Neighbors: 97.04545454545455
```

```
score=cross_val_score(Knn,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9804545454545457
```

NAIVE BAYES CLASSIFIER

```
from sklearn.naive_bayes import GaussianNB
NB=GaussianNB()
NB.fit(X_train,Y_train)
Y_Pred=NB.predict(X_test)

a=metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("NB")
print("Accuracy acquired from Naive Bayes:",a*100)
```

```
Accuracy acquired from Naive Bayes: 99.0909090909091
```

```
score=cross_val_score(NB,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9950000000000001
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
LR=LogisticRegression(random_state=2)
LR.fit(X_train,Y_train)
```

```
Y_Pred=LR.predict(X_test)

a=metrics.accuracy_score(Y_test,Y_Pred)
acc.append(a)
model.append("LR")
print("Accuracy acquired from Logistic Regression:",a*100)
```

```
Accuracy acquired from Logistic Regression: 95.22727272727273
```

```
score=cross_val_score(LR,features,target,cv=10,scoring="accuracy").mean()
score
```

```
0.9604545454545453
```

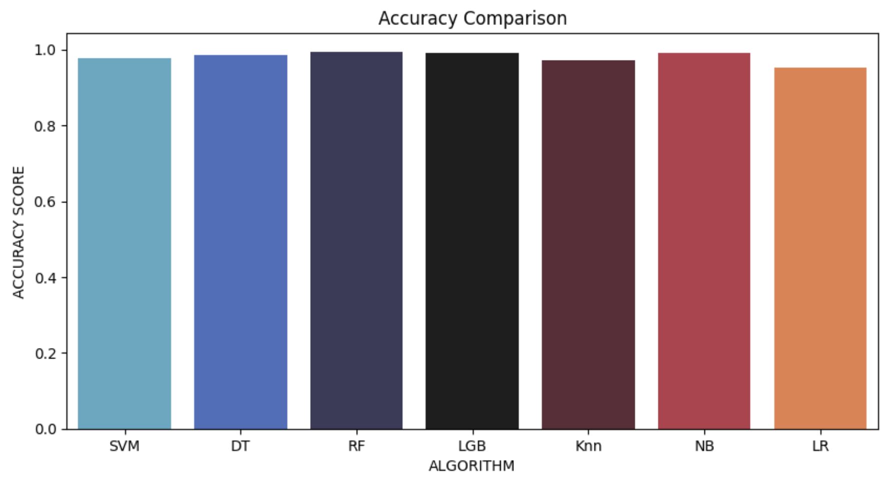
BAR CODE REPRESENTATION OF THE ACCURACY SCORES ACQUIRED FROM VARIOUS ALGORITHMS

```
accuracy_models = dict(zip(model,acc))
for a,b in accuracy_models.items():
    print (a,'-',b)
```

```
SVM - 0.9772727272727273
DT - 0.9840909090909091
RF - 0.9931818181818182
LGB - 0.990909090909091
Knn - 0.9704545454545455
NB - 0.990909090909091
LR - 0.9522727272727273
```

```
plt.figure(figsize=[10,5],dpi=100)
plt.title("Accuracy Comparison")
plt.xlabel("ALGORITHM")
plt.ylabel("ACCURACY SCORE")
sns.barplot(x=model,y=acc,palette="icefire")
```

```
<AxesSubplot:title={'center':'Accuracy Comparison'}, xlabel='ALGORITHM',
ylabel='ACCURACY SCORE'>
```



CLASSIFICATION REPORT AND CONFUSION MATRIX DISPLAY

```
lst=[SVM,DT,RF,LGB,Knn,NB,LR]
from sklearn.metrics import classification_report
for i in lst:
  i.fit(X_train,Y_train)
  Y_Pred=i.predict(X_test)
print(classification_report(Y_test,Y_Pred))
```

```
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00        13
      banana       1.00      1.00      1.00        17
   blackgram       0.86      0.75      0.80        16
    chickpea       1.00      1.00      1.00        21
     coconut       1.00      1.00      1.00        21
      coffee       1.00      1.00      1.00        22
      cotton       0.86      0.90      0.88        20
      grapes       1.00      1.00      1.00        18
        jute       0.84      0.93      0.88        28
 kidneybeans       1.00      1.00      1.00        14
      lentil       0.88      1.00      0.94        23
       maize       0.90      0.86      0.88        21
       mango       0.96      1.00      0.98        26
   mothbeans       0.84      0.84      0.84        19
    mungbean       1.00      0.96      0.98        24
   muskmelon       1.00      1.00      1.00        23
      orange       1.00      1.00      1.00        29
      papaya       1.00      0.95      0.97        19
  pigeonpeas       1.00      1.00      1.00        18
 pomegranate       1.00      1.00      1.00        17
        rice       0.85      0.69      0.76        16
  watermelon       1.00      1.00      1.00        15

    accuracy                           0.95       440
   macro avg       0.95      0.95      0.95       440
weighted avg       0.95      0.95      0.95       440
```

```
from sklearn.metrics import confusion_matrix
lst=[SVM,DT,RF,LGB,Knn,NB,LR]
for i in lst:
    i.fit(X_train,Y_train)
    Y_Pred=i.predict(X_test)
cm=confusion_matrix(Y_test,Y_Pred)
plt.figure(figsize=(15,15))
sns.heatmap(cm,annot=True,linewidths=.10,square=True,cmap ='crest');
```

```
plt.ylabel("Actual label");
plt.xlabel("Predicted label");
all_sample_title="Confusion Matrix"
plt.title(all_sample_title,size=30);
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-d026e5d78821> in <module>
      1 from sklearn.metrics import confusion_matrix
----> 2 lst=[SVM,DT,RF,LGB,Knn,NB,LR]
      3 for i in lst:
      4     i.fit(X_train,Y_train)
      5     Y_Pred=i.predict(X_test)

NameError: name 'SVM' is not defined
```

SEARCH STACK OVERFLOW

RANDOM PREDICTION

```
newdata=DT.predict([[83,45,60,28,70.3,7.0,150.9]])
print("RECOMMENDED CROP",newdata)
```

```
RECOMMENDED CROP ['papaya']
```

```
newdata=SVM.predict([[104,18,30,23.603016,60.3,6.7,140.91]])
print("RECOMMENDED CROP",newdata)
```

```
RECOMMENDED CROP ['coffee']
```

```
newdata=RF.predict([[104,18,30,23.603016,60.3,6.7,140.91]])
print("RECOMMENDED CROP",newdata)
```

```
RECOMMENDED CROP ['coffee']
```

```
newdata=LR.predict([[60,55,44,23.004459,82.320763,7.840207,263.964248]])
print("RECOMMENDED CROP",newdata)
```

```
RECOMMENDED CROP ['rice']
```