APPENDIX

MODULE 1 - Data Validation Process

```
#import libraries for access and functional purpose
import pandas as p
import numpy as n
import matplotlib.pyplot as plt
import seaborn as s

#read the given dataset
df = p.read_csv("df.csv")

df.head()

listcrops = p.Categorical(df['Crop'])
listcrops

df['Crop'].value_counts()

df['Crop'].nunique()

df['Crop'].unique()

df.shape

#To describe the dataframe
df.describe()

#Checking datatype and information about dataset
df.info()

df[df.dtypes[df.dtypes == 'float64'].index].describe()

p.Categorical(df['State_Name']).describe()

p.Categorical(df['District_Name']).describe()

p.Categorical(df['Mean Temp']).describe()

p.Categorical(df['Season']).describe()

p.Categorical(df['Average Humidity']).describe()

p.Categorical(df['rainfall']).describe()

p.Categorical(df['Crop']).describe()

#Checking for duplicate data
df.duplicated()

#find sum of duplicate data
sum(df.duplicated())
```

```
df.nunique()

#Correlation
df.corr()

#Checking minimum or maximum yields (100kg/2.47 acre)
print("Minimum yield of crops is  (100kg/2.47 acre):", df["Yield (Quintal/ Hectare) "].min())
print("Maximun yield of crops is  (100kg/2.47 acre):", df["Yield (Quintal/ Hectare) "].max())

#Checking minimum or maximum cost production for c2 scheme (per 2.47 acre)
print("Minimum cost production for c2 scheme(per 2.47 acre):", df["Cost of Production (`/Quintal) C2"].min())
print("Maximun cost production for c2 scheme(per 2.47 acre):", df["Cost of Production (`/Quintal) C2"].max())

#Rename the data
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)
#show the dataframe
df.head()
```

MODULE 2 - Data Pre-Processing with unvariate, bivariate, multivariate analysis

```
p.crosstab(df.State_Name,df.Crop)

df.rename(columns={'Mean Temp':'T'}, inplace=True)
df.rename(columns={'Average Humidity':'H'}, inplace=True)
df.rename(columns={'rainfall':'R'}, inplace=True)

#Rename the data
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)

p.crosstab(df.CC,df.CP)

df.rename(columns={'cost of production per yield':'CPPY'}, inplace=True)

p.crosstab(df.CC,df.CPPY)

p.crosstab(df.Crop,df.H)

df.dropna()

df['Y'].unique()

df.columns

from sklearn.preprocessing import LabelEncoder
var_mod = ['Y']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)
```

```
df['Y'].unique()

df['YPr']= df.Y.map({'13':0, '7':0, '11':0, '4':0, '23':0, '39':1, '10':0, '18':0, '36':1, '47':1, '8':0, '3':0,
      '38':1, '46':1, '5':0, '9':0, '2':0, '44':1, '17':0, '41':1, '6':0, '16':0, '35':1, '19':0,
      '0':0, '43':1, '12':0, '45':1, '25':0, '33':1, '29':0, '37':1, '32':1, '21':0, '42':1,
      '48':1, '30':1, '34':1, '26':0, '20':0, '31':1, '24':0, '27':0, '22':0, '40':1, '28':0,
      '1':0, '14':0, '15':0})


df['CPPY'].unique()

from sklearn.preprocessing import LabelEncoder
var_mod = ['CPPY']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)


df['CPPY'].unique()

df['CPPYPr']= df.CPPY.map({'126':0, '72':0, '200':0, '78':0, '144':0, '212':0, '149':0, '151':0, '178':0, '312':0,
      '187':0, '69':0, '205':0, '276':1, '76':0, '134':0, '66':0, '254':1, '150':0, '291':1,
      '155':0, '106':0, '87':0, '119':0, '183':0, '170':0, '28':0, '265':1, '180':0, '36':0,
      '293':1, '326':1, '148':0, '234':1, '3':0, '124':0, '111':0, '182':0, '166':0, '169':0,
      '270':1, '243':1, '98':0, '230':0, '282':1, '334':1, '188':0, '65':0, '34':0, '328':1,
      '160':0, '115':0, '89':0, '152':0, '240':1, '141':0, '233':1, '32':0, '281':1, '335':1,
      '109':0, '47':0, '136':0, '112':0, '261':1, '229':0, '42':0, '325':1, '213':0, '175':0,
      '125':0, '196':0, '292':1, '82':0, '235':1, '79':0, '105':0, '123':0, '210':0, '39':0,
      '146':0, '185':0, '201':0, '41':0, '164':0, '184':0, '222':0, '250':0, '301':1,
      '218':0, '217':0, '168':0, '264':1, '46':0, '103':0, '284':1, '198':0, '56':0, '171':0,
      '331':1, '256':1, '99':0, '58':0, '247':1, '286':1, '67':0, '133':0, '143':0, '204':0,
      '227':0, '194':0, '6':0, '280':1, '225':0, '48':0, '258':1, '94':0, '244':1, '294':1,
      '215':0, '132':0, '277':1, '71':0, '219':0, '315':1, '97':0, '91':0, '156':0, '289':1,
      '100':0, '295':1, '147':0, '214':0, '19':0, '223':0, '90':0, '269':1, '300':1, '114':0,
      '135':0, '173':0, '55':0, '113':0, '127':0, '73':0, '177':0, '274':1, '118':0, '191':0,
      '145':0, '307':1, '162':0, '228':0, '50':0, '5':0, '248':1, '203':0, '61':0, '129':0,
      '192':0, '83':0, '158':0, '206':0, '242':1, '267':1, '137':0, '92':0, '176':0, '298':1,
      '25':0, '296':1, '186':0, '239':1, '193':0, '165':0, '310':1, '20':0, '287':1, '75':0,
      '107':0, '271':1, '138':0, '121':0, '241':1, '74':0, '43':0, '232':0, '154':0, '181':0,
      '195':0, '102':0, '237':1, '110':0, '268':1, '49':0, '104':0, '64':0, '318':1, '13':0,
      '128':0, '153':0, '31':0, '93':0, '309':1, '257':0, '52':0, '262':1, '202':0, '174':0,
      '101':0, '224':0, '86':0, '45':0, '263':1, '167':0, '17':0, '29':0, '30':0, '54':0,
      '303':1, '163':0, '251':1, '142':0, '273':1, '96':0, '35':0, '327':1, '53':0, '15':0,
      '190':0, '308':1, '226':0, '139':0, '62':0, '2':0, '1':0, '27':0, '84':0, '285':1,
      '246':1, '23':0, '189':0, '299':1, '231':0, '24':0, '16':0, '333':1, '306':1, '18':0,
      '288':1, '199':0, '260':1, '323':1, '37':0, '278':1, '324':1, '140':0, '4':0, '245':1,
      '322':1, '329':1, '159':0, '80':0, '197':0, '275':1, '11':0, '40':0, '33':0, '0':0,
      '320':1, '290':1, '68':0, '220':0, '21':0, '330':1, '12':0, '157':0, '302':1, '44':0,
      '221':0, '216':0, '279':1, '63':0, '313':1, '311':1, '332':1, '266':1, '238':1,
      '211':0, '172':0, '14':0, '117':0, '161':0, '122':0, '60':0, '321':1, '95':0, '208':0,
      '272':1, '131':0, '207':0, '81':0, '314':1, '51':0, '283':1, '120':0, '38':0, '9':0,
      '130':0, '70':0, '179':0, '7':0, '10':0, '108':0, '255':1, '77':0, '88':0, '22':0,
      '57':0, '85':0, '304':1, '253':1, '249':0, '297':1, '116':0, '305':1, '317':1, '252':1,
      '236':1, '209':0, '259':1, '26':0, '319':1, '316':1, '59':0, '336':1, '8':0})
```

```
df.head()

df['YPr'].unique()

df['CPPYPr'].unique()

# Spliting Train/Test:

#preprocessing, split test and dataset, split response variable
X = df.drop(labels='CPPY', axis=1)
#Response variable
y = df.loc[:,'CPPY']

#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
print("Number of training dataset: ", len(X_train))
print("Number of testing dataset: ", len(X_test))
print("Total number of dataset: ", len(X_train)+len(X_test))

count_classes = p.value_counts(df['Crop'], sort = True).sort_index()
count_classes.plot(kind = 'bar', figsize=(20,15))
plt.title("Crop details")
plt.xlabel("Catogeries")
plt.ylabel("Strength values")
no=sum(df['CPPYPr']==0)
yes=sum(df['CPPYPr']==1)
colors=['orange','black']
locations=[1,2]
heights=[no,yes]
labels=['Unexpected Cost Production','Expected Cost Production']
plt.bar(locations,heights,color=colors,tick_label=labels,alpha=0.7)
plt.xlabel('Yield of Crost Production')
plt.ylabel('No. of each crop')
plt.title('Prediction results expecting from farmer by yield of crost production amount')
no=sum(df['YPr']==0)
yes=sum(df['YPr']==1)
colors=['orange','black']
locations=[1,2]
heights=[no,yes]
labels=['No Yield','Yield']
plt.bar(locations,heights,color=colors,tick_label=labels,alpha=0.7)
plt.xlabel('Yield of Crop')
plt.ylabel('No. of each crop')
plt.title('Prediction results expecting from farmer by yield of crop')
```

MODULE 3 - Data Visualization Process

```
#import libraries for access and functional purpose
import pandas as p
import numpy as n
import matplotlib.pyplot as plt
import seaborn as s
```

```
#read the given dataset
df = p.read_csv("df.csv")

df.rename(columns={'Mean Temp':'T'}, inplace=True)
df.rename(columns={'Average Humidity':'H'}, inplace=True)
df.rename(columns={'rainfall':'R'}, inplace=True)
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)

df.rename(columns={'cost of production per yield':'CPPY'}, inplace=True)

df['CP'].hist(figsize=(7,6), color='orange', alpha=0.7)
plt.xlabel('Cost Production (per 100kg)')
plt.ylabel('No of counts')
plt.title('Cost Production per 100kg by counts')

df.columns

df['CPPY'].hist(figsize=(7,6), color='black', alpha=0.7)
plt.xlabel('Cost Production of crop')
plt.ylabel('No of counts')
plt.title('Cost Production of crop by counts')

# Heatmap plot diagram
fig, ax = plt.subplots(figsize=(15,10))
s.heatmap(df.corr(), ax=ax, annot=True)

df.columns

df.boxplot(column="CP", by="Season", figsize=(15,10))

df.boxplot(column="CPPY", by="Season", figsize=(15,10))

#Propagation by variable
def PropByVar(df, variable):
    dataframe_pie = df[variable].value_counts()
    ax = dataframe_pie.plot.pie(figsize=(10,10), autopct='%1.2f%%', fontsize = 12)
    ax.set_title(variable + ' (in Percentage)', fontsize = 15)
    return n.round(dataframe_pie/df.shape[0]*100,2)

PropByVar(df, 'Crop')

#Propagation by variable
def PropByVar(df, variable):
    dataframe_pie = df[variable].value_counts()
    ax = dataframe_pie.plot.pie(figsize=(10,10), autopct='%1.2f%%', fontsize = 12)
    ax.set_title(variable + ' (in Percentage)', fontsize = 15)
    return n.round(dataframe_pie/df.shape[0]*100,2)


PropByVar(df, 'State_Name')

#Propagation by variable
def PropByVar(df, variable):
    dataframe_pie = df[variable].value_counts()
    ax = dataframe_pie.plot.pie(figsize=(10,10), autopct='%1.2f%%', fontsize = 12)
```

```
    ax.set_title(variable + ' (in Percentage)', fontsize = 15)
    return n.round(dataframe_pie/df.shape[0]*100,2)


PropByVar(df, 'CPPY')

#Propagation by variable
def PropByVar(df, variable):
    dataframe_pie = df[variable].value_counts()
    ax = dataframe_pie.plot.pie(figsize=(10,10), autopct='%1.2f%%', fontsize = 12)
    ax.set_title(variable + ' (in Percentage)', fontsize = 15)
    return n.round(dataframe_pie/df.shape[0]*100,2)


PropByVar(df, 'Y')

count_classes = p.value_counts(df['State_Name'], sort = True).sort_index()
count_classes.plot(kind = 'bar', figsize=(20,15))
plt.title("Dataset by each states")
plt.xlabel("Number of States")
plt.ylabel("Given data counts")

#Density Plots
plt = df.plot(kind= 'density', subplots=True, layout=(4,3), sharex=False,
                  sharey=False,fontsize=12, figsize=(15,10))
```

MODULE 4 – Outlier detection process

```
#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n
#read the given dataset
df = p.read_csv("df.csv")

df.rename(columns={'Mean Temp':'T'}, inplace=True)
df.rename(columns={'Average Humidity':'H'}, inplace=True)
df.rename(columns={'rainfall':'R'}, inplace=True)
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)
df.rename(columns={'cost of production per yield':'CPPY'}, inplace=True)

from sklearn.preprocessing import LabelEncoder
var_mod = ['Y','CPPY']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)


df['YPr']= df.Y.map({'13':0, '7':0, '11':0, '4':0, '23':0, '39':1, '10':0, '18':0, '36':1, '47':1, '8':0, '3':0,
      '38':1, '46':1, '5':0, '9':0, '2':0, '44':1, '17':0, '41':1, '6':0, '16':0, '35':1, '19':0,
      '0':0, '43':1, '12':0, '45':1, '25':0, '33':1, '29':0, '37':1, '32':1, '21':0, '42':1,
      '48':1, '30':1, '34':1, '26':0, '20':0, '31':1, '24':0, '27':0, '22':0, '40':1, '28':0,
      '1':0, '14':0, '15':0})
```

```
df['CPPYPr']= df.CPPY.map({'126':0, '72':0, '200':0, '78':0, '144':0, '212':0, '149':0, '151':0, '178':0, '312':0,
      '187':0, '69':0, '205':0, '276':1, '76':0, '134':0, '66':0, '254':1, '150':0, '291':1,
      '155':0, '106':0, '87':0, '119':0, '183':0, '170':0, '28':0, '265':1, '180':0, '36':0,
      '293':1, '326':1, '148':0, '234':1, '3':0, '124':0, '111':0, '182':0, '166':0, '169':0,
      '270':1, '243':1, '98':0, '230':0, '282':1, '334':1, '188':0, '65':0, '34':0, '328':1,
      '160':0, '115':0, '89':0, '152':0, '240':1, '141':0, '233':1, '32':0, '281':1, '335':1,
      '109':0, '47':0, '136':0, '112':0, '261':1, '229':0, '42':0, '325':1, '213':0, '175':0,
      '125':0, '196':0, '292':1, '82':0, '235':1, '79':0, '105':0, '123':0, '210':0, '39':0,
      '146':0, '185':0, '201':0, '41':0, '164':0, '184':0, '222':0, '250':0, '301':1,
      '218':0, '217':0, '168':0, '264':1, '46':0, '103':0, '284':1, '198':0, '56':0, '171':0,
      '331':1, '256':1, '99':0, '58':0, '247':1, '286':1, '67':0, '133':0, '143':0, '204':0,
      '227':0, '194':0, '6':0, '280':1, '225':0, '48':0, '258':1, '94':0, '244':1, '294':1,
      '215':0, '132':0, '277':1, '71':0, '219':0, '315':1, '97':0, '91':0, '156':0, '289':1,
      '100':0, '295':1, '147':0, '214':0, '19':0, '223':0, '90':0, '269':1, '300':1, '114':0,
      '135':0, '173':0, '55':0, '113':0, '127':0, '73':0, '177':0, '274':1, '118':0, '191':0,
      '145':0, '307':1, '162':0, '228':0, '50':0, '5':0, '248':1, '203':0, '61':0, '129':0,
      '192':0, '83':0, '158':0, '206':0, '242':1, '267':1, '137':0, '92':0, '176':0, '298':1,
      '25':0, '296':1, '186':0, '239':1, '193':0, '165':0, '310':1, '20':0, '287':1, '75':0,
      '107':0, '271':1, '138':0, '121':0, '241':1, '74':0, '43':0, '232':0, '154':0, '181':0,
      '195':0, '102':0, '237':1, '110':0, '268':1, '49':0, '104':0, '64':0, '318':1, '13':0,
      '128':0, '153':0, '31':0, '93':0, '309':1, '257':0, '52':0, '262':1, '202':0, '174':0,
      '101':0, '224':0, '86':0, '45':0, '263':1, '167':0, '17':0, '29':0, '30':0, '54':0,
      '303':1, '163':0, '251':1, '142':0, '273':1, '96':0, '35':0, '327':1, '53':0, '15':0,
      '190':0, '308':1, '226':0, '139':0, '62':0, '2':0, '1':0, '27':0, '84':0, '285':1,
      '246':1, '23':0, '189':0, '299':1, '231':0, '24':0, '16':0, '333':1, '306':1, '18':0,
      '288':1, '199':0, '260':1, '323':1, '37':0, '278':1, '324':1, '140':0, '4':0, '245':1,
      '322':1, '329':1, '159':0, '80':0, '197':0, '275':1, '11':0, '40':0, '33':0, '0':0,
      '320':1, '290':1, '68':0, '220':0, '21':0, '330':1, '12':0, '157':0, '302':1, '44':0,
      '221':0, '216':0, '279':1, '63':0, '313':1, '311':1, '332':1, '266':1, '238':1,
      '211':0, '172':0, '14':0, '117':0, '161':0, '122':0, '60':0, '321':1, '95':0, '208':0,
      '272':1, '131':0, '207':0, '81':0, '314':1, '51':0, '283':1, '120':0, '38':0, '9':0,
      '130':0, '70':0, '179':0, '7':0, '10':0, '108':0, '255':1, '77':0, '88':0, '22':0,
      '57':0, '85':0, '304':1, '253':1, '249':0, '297':1, '116':0, '305':1, '317':1, '252':1,
      '236':1, '209':0, '259':1, '26':0, '319':1, '316':1, '59':0, '336':1, '8':0})

df.columns

df.pivot_table(values='YPr',index = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area',
      'R', 'H', 'T', 'CC', 'CP', 'Y'])

df.pivot_table(values='CPPYPr',index = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop',
'Area',
      'R', 'H', 'T', 'CC', 'CP', 'Y'])

df['State_Name'].unique()

TN = df[df.State_Name.str.contains('Tamil Nadu')]
TN.Crop.nunique()

TN.Crop.unique()

TN.pivot_table(values='CPPYPr',index = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop',
'Area',
      'R', 'H', 'T', 'CC', 'CP', 'Y'])

TN.pivot_table(values='YPr',index = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area',
```

```
      'R', 'H', 'T', 'CC', 'CP', 'Y'])

def y_No_y_bar_plot(df, bygroup):
    dataframe_by_Group = p.crosstab(df[bygroup], columns=df["YPr"], normalize = 'index')
    dataframe_by_Group = n.round((dataframe_by_Group * 100), decimals=2)
    ax = dataframe_by_Group.plot.bar(figsize=(10,5));
    vals = ax.get_yticks()
    ax.set_yticklabels(['{:3.0f}%'.format(x) for x in vals]);
    ax.set_xticklabels(dataframe_by_Group.index,rotation = 0, fontsize = 15);
    ax.set_title('Crop Yield Prediction Vs No Crop Yield Prediction (%) (by ' +
dataframe_by_Group.index.name + ')\n', fontsize = 15)
    ax.set_xlabel(dataframe_by_Group.index.name, fontsize = 12)
    ax.set_ylabel('(%)', fontsize = 12)
    ax.legend(loc = 'upper left',bbox_to_anchor=(1.0,1.0), fontsize= 12)
    rects = ax.patches

    # Add Data Labels

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2,
                height + 2,
                str(height)+'%',
                ha='center',
                va='bottom',
                fontsize = 12)
    return dataframe_by_Group


y_No_y_bar_plot(df, 'Season')


def c_No_y_bar_plot(df, bygroup):
    dataframe_by_Group = p.crosstab(df[bygroup], columns=df["CPPYPr"], normalize = 'index')
    dataframe_by_Group = n.round((dataframe_by_Group * 100), decimals=2)
    ax = dataframe_by_Group.plot.bar(figsize=(10,5));
    vals = ax.get_yticks()
    ax.set_yticklabels(['{:3.0f}%'.format(x) for x in vals]);
    ax.set_xticklabels(dataframe_by_Group.index,rotation = 0, fontsize = 15);
    ax.set_title('Crop Yield Production cost Vs No Crop Yield Production cost (%) (by ' +
dataframe_by_Group.index.name + ')\n', fontsize = 15)
    ax.set_xlabel(dataframe_by_Group.index.name, fontsize = 12)
    ax.set_ylabel('(%)', fontsize = 12)
    ax.legend(loc = 'upper left',bbox_to_anchor=(1.0,1.0), fontsize= 12)
    rects = ax.patches

    # Add Data Labels

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2,
                height + 2,
                str(height)+'%',
                ha='center',
                va='bottom',
                fontsize = 12)
    return dataframe_by_Group
```

```
c_No_y_bar_plot(df, 'Season')

df.columns
c_No_y_bar_plot(df, 'District_Name')

y_No_y_bar_plot(df, 'District_Name')
```

MODULE 5 - Comparing best accuracy and entropy

```
#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n
#read the given dataset
df = p.read_csv("df.csv")

df.rename(columns={'Mean Temp':'T'}, inplace=True)
df.rename(columns={'Average Humidity':'H'}, inplace=True)
df.rename(columns={'rainfall':'R'}, inplace=True)
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)
df.rename(columns={'cost of production per yield':'CPPY'}, inplace=True)

from sklearn.preprocessing import LabelEncoder
var_mod = ['Y']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)
df['YPr']= df.Y.map({'13':0, '7':0, '11':0, '4':0, '23':0, '39':1, '10':0, '18':0, '36':1, '47':1, '8':0, '3':0,
       '38':1, '46':1, '5':0, '9':0, '2':0, '44':1, '17':0, '41':1, '6':0, '16':0, '35':1, '19':0,
       '0':0, '43':1, '12':0, '45':1, '25':0, '33':1, '29':0, '37':1, '32':1, '21':0, '42':1,
       '48':1, '30':1, '34':1, '26':0, '20':0, '31':1, '24':0, '27':0, '22':0, '40':1, '28':0,
       '1':0, '14':0, '15':0})



df.columns

from sklearn.preprocessing import LabelEncoder
var_mod = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area',
       'R', 'H', 'T', 'CC', 'CP', 'Y','CPPY']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)
df.head()

#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.
from sklearn.metrics import confusion_matrix, classification_report, matthews_corrcoef, cohen_kappa_score, accuracy_score, average_precision_score, roc_auc_score

# Prediction of Crop by yield
```

```
X = df.drop(labels='YPr', axis=1)
#Response variable
y = df.loc[:,'YPr']

del df
#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)

#for our convienient we delete X,y variable for differentiate confusion
del X, y

Logistic Regression

from sklearn.linear_model import LogisticRegression
logR= LogisticRegression()

logR.fit(X_train,y_train)

predictR = logR.predict(X_test)
print(classification_report(y_test,predictR))
x = (accuracy_score(y_test,predictR)*100)


print('Accuracy result is', x)
print("")

print("")
print(confusion_matrix(y_test,predictR))

Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()

dtree.fit(X_train, y_train)

predictDT = dtree.predict(X_test)
print(classification_report(y_test,predictDT))
x = (accuracy_score(y_test,predictDT)*100)



print('Accuracy result is', x)
print("")
print(confusion_matrix(y_test,predictDT))


RandomForest Classifier

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

rf.fit(X_train, y_train)
```

```
predictrf = rf.predict(X_test)


print(classification_report(y_test,predictrf
                            ))
x = (accuracy_score(y_test,predictrf)*100)



print('Accuracy result is', x)

print("")
print(confusion_matrix(y_test,predictrf))
```

K-Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn.fit(X_train, y_train)

predictknn = knn.predict(X_test)

print(classification_report(y_test,predictknn
                            ))
x = (accuracy_score(y_test,predictknn)*100)



print('Accuracy result is', x)

print("")

print(confusion_matrix(y_test,predictknn))
```

Support Vector Classifier

```
from sklearn.svm import SVC
s = SVC()

s.fit(X_train, y_train)

predictSV = s.predict(X_test)

print(classification_report(y_test,predictSV
                            ))
x = (accuracy_score(y_test,predictSV)*100)


print('Accuracy result is', x)



print("")
print(confusion_matrix(y_test,predictSV))
```

```
# Prediction of Crop by Cost production

#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n
#read the given dataset
df = p.read_csv("df.csv")

df.rename(columns={'Mean Temp':'T'}, inplace=True)
df.rename(columns={'Average Humidity':'H'}, inplace=True)
df.rename(columns={'rainfall':'R'}, inplace=True)
df.rename(columns={'Cost of Cultivation (`/Hectare) C2':'CC'}, inplace=True)
df.rename(columns={'Cost of Production (`/Quintal) C2':'CP'}, inplace=True)
df.rename(columns={'Yield (Quintal/ Hectare) ':'Y'}, inplace=True)
df.rename(columns={'cost of production per yield':'CPPY'}, inplace=True)

from sklearn.preprocessing import LabelEncoder
var_mod = ['CPPY']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)
df['CPPYPr']= df.CPPY.map({'126':0, '72':0, '200':0, '78':0, '144':0, '212':0, '149':0, '151':0, '178':0, '312':0,
       '187':0, '69':0, '205':0, '276':1, '76':0, '134':0, '66':0, '254':1, '150':0, '291':1,
       '155':0, '106':0, '87':0, '119':0, '183':0, '170':0, '28':0, '265':1, '180':0, '36':0,
       '293':1, '326':1, '148':0, '234':1, '3':0, '124':0, '111':0, '182':0, '166':0, '169':0,
       '270':1, '243':1, '98':0, '230':0, '282':1, '334':1, '188':0, '65':0, '34':0, '328':1,
       '160':0, '115':0, '89':0, '152':0, '240':1, '141':0, '233':1, '32':0, '281':1, '335':1,
       '109':0, '47':0, '136':0, '112':0, '261':1, '229':0, '42':0, '325':1, '213':0, '175':0,
       '125':0, '196':0, '292':1, '82':0, '235':1, '79':0, '105':0, '123':0, '210':0, '39':0,
       '146':0, '185':0, '201':0, '41':0, '164':0, '184':0, '222':0, '250':0, '301':1,
       '218':0, '217':0, '168':0, '264':1, '46':0, '103':0, '284':1, '198':0, '56':0, '171':0,
       '331':1, '256':1, '99':0, '58':0, '247':1, '286':1, '67':0, '133':0, '143':0, '204':0,
       '227':0, '194':0, '6':0, '280':1, '225':0, '48':0, '258':1, '94':0, '244':1, '294':1,
       '215':0, '132':0, '277':1, '71':0, '219':0, '315':1, '97':0, '91':0, '156':0, '289':1,
       '100':0, '295':1, '147':0, '214':0, '19':0, '223':0, '90':0, '269':1, '300':1, '114':0,
       '135':0, '173':0, '55':0, '113':0, '127':0, '73':0, '177':0, '274':1, '118':0, '191':0,
       '145':0, '307':1, '162':0, '228':0, '50':0, '5':0, '248':1, '203':0, '61':0, '129':0,
       '192':0, '83':0, '158':0, '206':0, '242':1, '267':1, '137':0, '92':0, '176':0, '298':1,
       '25':0, '296':1, '186':0, '239':1, '193':0, '165':0, '310':1, '20':0, '287':1, '75':0,
       '107':0, '271':1, '138':0, '121':0, '241':1, '74':0, '43':0, '232':0, '154':0, '181':0,
       '195':0, '102':0, '237':1, '110':0, '268':1, '49':0, '104':0, '64':0, '318':1, '13':0,
       '128':0, '153':0, '31':0, '93':0, '309':1, '257':0, '52':0, '262':1, '202':0, '174':0,
       '101':0, '224':0, '86':0, '45':0, '263':1, '167':0, '17':0, '29':0, '30':0, '54':0,
       '303':1, '163':0, '251':1, '142':0, '273':1, '96':0, '35':0, '327':1, '53':0, '15':0,
       '190':0, '308':1, '226':0, '139':0, '62':0, '2':0, '1':0, '27':0, '84':0, '285':1,
       '246':1, '23':0, '189':0, '299':1, '231':0, '24':0, '16':0, '333':1, '306':1, '18':0,
       '288':1, '199':0, '260':1, '323':1, '37':0, '278':1, '324':1, '140':0, '4':0, '245':1,
       '322':1, '329':1, '159':0, '80':0, '197':0, '275':1, '11':0, '40':0, '33':0, '0':0,
       '320':1, '290':1, '68':0, '220':0, '21':0, '330':1, '12':0, '157':0, '302':1, '44':0,
       '221':0, '216':0, '279':1, '63':0, '313':1, '311':1, '332':1, '266':1, '238':1,
       '211':0, '172':0, '14':0, '117':0, '161':0, '122':0, '60':0, '321':1, '95':0, '208':0,
       '272':1, '131':0, '207':0, '81':0, '314':1, '51':0, '283':1, '120':0, '38':0, '9':0,
       '130':0, '70':0, '179':0, '7':0, '10':0, '108':0, '255':1, '77':0, '88':0, '22':0,
       '57':0, '85':0, '304':1, '253':1, '249':0, '297':1, '116':0, '305':1, '317':1, '252':1,
```

```
      '236':1, '209':0, '259':1, '26':0, '319':1, '316':1, '59':0, '336':1, '8':0})



from sklearn.preprocessing import LabelEncoder
var_mod = ['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area',
      'R', 'H', 'T', 'CC', 'CP', 'Y','CPPY']
le = LabelEncoder()
for i in var_mod:
   df[i] = le.fit_transform(df[i]).astype(str)
df.head()

#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.
from sklearn.metrics import confusion_matrix, classification_report, matthews_corrcoef, cohen_kappa_score, accuracy_score, average_precision_score, roc_auc_score

X = df.drop(labels='CPPYPr', axis=1)
#Response variable
y = df.loc[:,'CPPYPr']

del df
#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)

#for our convienient we delete X,y variable for differentiate confusion
del X, y

Logistic Regression

from sklearn.linear_model import LogisticRegression
logR= LogisticRegression()

logR.fit(X_train,y_train)

predictR = logR.predict(X_test)
print(classification_report(y_test,predictR))
x = (accuracy_score(y_test,predictR)*100)


print('Accuracy result is', x)
print("")

print("")
print(confusion_matrix(y_test,predictR))

Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()

dtree.fit(X_train, y_train)

predictDT = dtree.predict(X_test)
print(classification_report(y_test,predictDT))
```

```
x = (accuracy_score(y_test,predictDT)*100)



print('Accuracy result is', x)
print("")
print(confusion_matrix(y_test,predictDT))
```

RandomForest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

predictrf = rf.predict(X_test)


print(classification_report(y_test,predictrf
                            ))
x = (accuracy_score(y_test,predictrf)*100)



print('Accuracy result is', x)

print("")
print(confusion_matrix(y_test,predictrf))
```

K-Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn.fit(X_train, y_train)

predictknn = knn.predict(X_test)

print(classification_report(y_test,predictknn
                            ))
x = (accuracy_score(y_test,predictknn)*100)



print('Accuracy result is', x)

print("")

print(confusion_matrix(y_test,predictknn))
```

Support Vector Classifier

```
from sklearn.svm import SVC
s = SVC()
```

```
s.fit(X_train, y_train)

predictSV = s.predict(X_test)

print(classification_report(y_test,predictSV
                    ))
x = (accuracy_score(y_test,predictSV)*100)


print('Accuracy result is', x)



print("")
print(confusion_matrix(y_test,predictSV))
```

MODULE – GUI

```
from tkinter import *
import numpy as np
import pandas as pd

df = pd.read_csv("re.csv")

df

df.shape

df.columns



l1=['Sugercane', 'Rice', 'Tobacco', 'Wheat', 'Coconut', 'Yam',
    'Sweetpotato', 'Tapioca', 'Turmeric', 'WaterMelon', 'Urad', 'Varagu',
    'Banana']

l2=['DINDIGUL', 'THE NILGIRIS', 'KARUR', 'KRISHNAGIRI', 'MADURAI',
    'NAGAPATTINAM', 'NAMAKKAL', 'PERAMBALUR', 'PUDUKKOTTAI', 'SALEM',
    'THANJAVUR', 'THENI', 'THIRUVARUR', 'TIRUCHIRAPPALLI', 'TIRUNELVELI',
    'TIRUPPUR', 'TIRUVANNAMALAI', 'VELLORE', 'VILLUPURAM', 'VIRUDHUNAGAR',
    'COIMBATORE', 'CUDDALORE', 'DHARMAPURI', 'ERODE', 'KANNIYAKUMARI',
    'KANCHIPURAM', 'ARIYALUR', 'RAMANATHAPURAM', 'SIVAGANGA', 'THIRUVALLUR',
    'TUTICORIN']



l4=[ 'Summer', 'Winter', 'Kharif', 'Rabi', 'Whole Year',
    'Autumn']

l5=['TA50', 'TA60', 'TA80']

l6=['HB25', 'HA25', 'HA60', 'HA80']

df['Class'].unique()
```

```
cropyc=['Cost Yield is ten tons and Cost Production is ten lakshs',
      'Cost Yield is one ton and Cost Production is twenty thousand',
      'Cost Yield is twenty tons and Cost Production is fourty lakshs',
      'Cost Yield is two tons and Cost Production is fourty thousand',
      'Cost Yield is three tons and Cost Production is sixty thousand',
      'Cost Yield is six tons and Cost Production is six lakshs ',
      'Cost Yield is two tons and Cost Production is fourty  thousand',
      'Cost Yield is five tons and Cost Production is one lakshs',
      'Cost Yield is ten tons and Cost Production is two lakshs']

l7=['Sugercane', 'Rice', 'Tobacco', 'Wheat', 'Coconut', 'Yam',
      'Sweetpotato', 'Tapioca', 'Turmeric', 'WaterMelon', 'Urad', 'Varagu',
      'Banana','DINDIGUL', 'THE NILGIRIS', 'KARUR', 'KRISHNAGIRI', 'MADURAI',
      'NAGAPATTINAM', 'NAMAKKAL', 'PERAMBALUR', 'PUDUKKOTTAI', 'SALEM',
      'THANJAVUR', 'THENI', 'THIRUVARUR', 'TIRUCHIRAPPALLI', 'TIRUNELVELI',
      'TIRUPPUR', 'TIRUVANNAMALAI', 'VELLORE', 'VILLUPURAM', 'VIRUDHUNAGAR',
      'COIMBATORE', 'CUDDALORE', 'DHARMAPURI', 'ERODE', 'KANNIYAKUMARI',
      'KANCHIPURAM', 'ARIYALUR', 'RAMANATHAPURAM', 'SIVAGANGA', 'THIRUVALLUR',
      'TUTICORIN','Summer', 'Winter', 'Kharif', 'Rabi', 'Whole Year',
      'Autumn','TA50', 'TA60', 'TA80','HB25', 'HA25', 'HA60', 'HA80']

l8=[]
for x in range(0,len(l7)):
   l8.append(0)


df.replace({'Class':{'Cost Yield is ten tons and Cost Production is ten lakshs':0,
      'Cost Yield is one ton and Cost Production is twenty thousand':1,
      'Cost Yield is twenty tons and Cost Production is fourty lakshs':2,
      'Cost Yield is two tons and Cost Production is fourty thousand':3,
      'Cost Yield is three tons and Cost Production is sixty thousand':4,
      'Cost Yield is six tons and Cost Production is six lakshs ':5,
      'Cost Yield is two tons and Cost Production is fourty  thousand':6,
      'Cost Yield is five tons and Cost Production is one lakshs':7,
      'Cost Yield is ten tons and Cost Production is two lakshs':8}},inplace=True)

X= df[l7]


y = df[["Class"]]
np.ravel(y)

# TRAINING DATA tr ----------------------------------------------------------------------------
tr=pd.read_csv("tere.csv")

X_test= tr[l7]
y_test = tr[["Class"]]
np.ravel(y_test)

def randomforest():
   from sklearn.ensemble import RandomForestClassifier
   clf4 = RandomForestClassifier()
   clf4 = clf4.fit(X,np.ravel(y))

   # calculating accuracy-------------------------------------------------------------------
   from sklearn.metrics import accuracy_score
   y_pred=clf4.predict(X_test)
```

```
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred,normalize=False))
    # ---------------------------------------------------

    terms = [Dist.get(),Season.get(),Tl.get(),Hl.get(),Crop.get()]

    for k in range(0,len(l7)):
        for z in terms:
            if(z==l7[k]):
                l8[k]=1

    inputtest = [l8]
    predict = clf4.predict(inputtest)
    predicted=predict[0]

    h='no'
    for a in range(0,len(cropyc)):
        if(predicted == a):
            h='yes'
            break

    if (h=='yes'):
        t2.delete("1.0", END)
        t2.insert(END, cropyc[a])
    else:
        t2.delete("1.0", END)
        t2.insert(END, "Not Found")


def LogisticRegression():
    from sklearn.linear_model import LogisticRegression
    gnb= LogisticRegression()

    gnb=gnb.fit(X,np.ravel(y))

    # calculating accuracy---------------------------------------------------------------------
    from sklearn.metrics import accuracy_score
    y_pred=gnb.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred,normalize=False))
    # ---------------------------------------------------

    terms = [Dist.get(),Season.get(),Tl.get(),Hl.get(),Crop.get()]
    for k in range(0,len(l7)):
        for z in terms:
            if(z==l7[k]):
                l8[k]=1

    inputtest = [l8]
    predict = gnb.predict(inputtest)
    predicted=predict[0]

    h='no'
    for a in range(0,len(cropyc)):
        if(predicted == a):
            h='yes'
            break
```

```
    if (h=='yes'):
        t3.delete("1.0", END)
        t3.insert(END, cropyc[a])
    else:
        t3.delete("1.0", END)
        t3.insert(END, "Not Found")


root = Tk()
root.configure(background='black')

# gui_stuff------------------------------------------------------------------------------------


# entry variables

Dist = StringVar()
Dist.set(None)

Season = StringVar()
Season.set(None)

Crop =StringVar()
Crop.set(None)

Tl = StringVar()
Tl.set(None)

Hl = StringVar()
Hl.set(None)

Yl = StringVar()
Yl.set(None)



fn= StringVar()
vn= StringVar()

rl= IntVar()

# Heading
w2 = Label(root, justify=LEFT, text="Prediction of Yield and Yield cost by Crop using Machine Learning",
fg="white", bg="red")
w2.config(font=("Elephant", 20))
w2.grid(row=1, column=0, columnspan=2, padx=100)
w2 = Label(root, justify=LEFT, text="Tamilnadu Agricultural Department", fg="green")
w2.config(font=("Times Roman", 15))
w2.grid(row=2, column=0, columnspan=2, padx=100)

# labels
FNLb = Label(root, text="Farmer Name",fg="yellow", bg="black")
FNLb.grid(row=5, column=0, pady=15, sticky=W)

VNLb = Label(root, text="Village Name",fg="yellow", bg="black")
VNLb.grid(row=6, column=0, pady=15, sticky=W)
```

```
RFLb = Label(root, text="Rainfall Value",fg="yellow", bg="black")
RFLb.grid(row=8, column=1, pady=15, sticky=W)



# labels
dtLb = Label(root, text="District Name",fg="yellow", bg="black")
dtLb.grid(row=8, column=0, pady=15, sticky=W)

seLb = Label(root, text="Season Details",fg="yellow", bg="black")
seLb.grid(row=9, column=0, pady=15, sticky=W)

tlLb = Label(root, text="Temperature Level",fg="yellow", bg="black")
tlLb.grid(row=5, column=1, pady=15, sticky=W)

hlLb = Label(root, text="Humidity Level",fg="yellow", bg="black")
hlLb.grid(row=6, column=1, pady=15, sticky=W)


crLb = Label(root, text="Crop Details",fg="yellow", bg="black")
crLb.grid(row=7, column=1, pady=15, sticky=W)

destreeLb = Label(root, text="LogisticRegression", fg="white", bg="red")
destreeLb.grid(row=14, column=0, pady=10,sticky=W)

ranfLb = Label(root, text="RandomForest", fg="white", bg="red")
ranfLb.grid(row=15, column=0, pady=10, sticky=W)


# entries
OPTIONS1 = sorted(l1)
OPTIONS2 = sorted(l2)

OPTIONS4 = sorted(l4)
OPTIONS5 = sorted(l5)
OPTIONS6 = sorted(l6)




FNEn = Entry(root, textvariable=fn)
FNEn.grid(row=5, column=0)

VNEn = Entry(root, textvariable=vn)
VNEn.grid(row=6, column=0)

RFEn = Entry(root, textvariable=rl)
RFEn.grid(row=8, column=1)

crEn = OptionMenu(root, Crop,*OPTIONS1)
```

```
crEn.grid(row=7, column=1)


dtEn = OptionMenu(root, Dist,*OPTIONS2)
dtEn.grid(row=8, column=0)

seEn = OptionMenu(root, Season,*OPTIONS4)
seEn.grid(row=9, column=0)

tlEn = OptionMenu(root, Tl,*OPTIONS5)
tlEn.grid(row=5, column=1)

hlEn = OptionMenu(root, Hl,*OPTIONS6)
hlEn.grid(row=6, column=1)

dst = Button(root, text="LogisticRegression Algorithm", command=DecisionTree,bg="cyan",fg="green")
dst.grid(row=14, column=1,padx=10)

rnf = Button(root, text="Randomforest Algorithm", command=randomforest,bg="cyan",fg="green")
rnf.grid(row=15, column=1,padx=10)


#textfileds
t1 = Text(root, height=1, width=70,bg="orange",fg="black")
t1.grid(row=14, column=0, padx=10)

t2 = Text(root, height=1, width=70,bg="orange",fg="black")
t2.grid(row=15, column=0 , padx=10)

root.mainloop()
```