INTELLIGENT CROP RECOMMENDATION AND YIELD PREDICTION SYSTEM USING ML

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input', FloatTensorType([None, 4]))]
onx = convert_sklearn(clf, initial_types = initial_type)
with open("crop_recommendation.onnx","wb") as f:
   f.write(onx.SerializeToString())
#importing the dataset
crop = pd.read_csv('/Crop_recommendation.csv')
crop.drop(crop[crop.label == 'muskmelon'].index, inplace = True )
#copying original data
data = crop.copy()
data.info()
#checking for null values
data.isnull().sum().any()
#checking for unique values
for i in data.columns:
   print("column Name : ",i.upper())
   print("No. of unique values : {} \n".format(data[i].nunique()))
   if(data[i].dtype == 'object'):
      print('Unique values : ',pd.unique(data[i]))
#label encoding for output variable
from sklearn.preprocessing import LabelEncoder
```

```
encod = LabelEncoder()
data['Encoded_label'] = encod.fit_transform(data.label) #label will be encoded in alphabetical order
#encoded labels for classes
a = pd.DataFrame(pd.unique(data.label));
a.rename(columns={0:'label'},inplace=True)
b = pd.DataFrame(pd.unique(data.Encoded_label));
b.rename(columns={0:'encoded'},inplace=True)
classes = pd.concat([a,b],axis=1).sort_values('encoded').set_index('label')
classes
#fetching the label for given encoded value
a=12
for i in range(0,len(classes)):
   if(classes.encoded[i]==a):
      print(classes.index[i].upper())
#dropping duplicate values
data = data.drop_duplicates()
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numerical columns for correlation
numerical_data = data.select_dtypes(include=['float64', 'int64'])

# Check the correlation and plot the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(numerical_data.corr(), annot=True, cmap='RdBu')

# Show the plot
plt.show()

#EDA
data.describe()
```

```
#checking for outliers in the data
for i in data.columns[:-2]:
    print('Variable Name :',i.upper())
    fig, axes = plt.subplots(1,2,figsize=(8,4))
    axes[0].set_title('Distribution')
    axes[1].set_title('Outliers Detection')
    data[i].hist(ax=axes[0])
    sns.boxplot(data[i],ax=axes[1])
    plt.show()
    print('\n')
#plotting effect of input variable with output variable
for i in data.columns[:-2]:
    plt.figure(figsize=(15,5))
    print('Variable :',i.upper())
    sns.boxplot(x=data.label,y=data[i])
    plt.grid()
    plt.xticks(rotation=90)
    plt.show()
#which crops can grow at higher temperature .i.e., temperature > 30
x = pd.DataFrame(pd.crosstab(data.label[data.temperature > 30],'count',normalize=True)*100)
x.plot.pie(y = 'count',autopct='%1.1f%%',figsize=(8,8),legend=None,shadow=True, startangle=90)
plt.title('Probability of crops grow when temperature > 30')
plt.show()

#which crops can grow at higher rainfall .i.e., rainfall > 150mm
x1 = pd.DataFrame(pd.crosstab(data.label[data.rainfall > 150],'count',normalize=True)*100)
x1.plot.pie(y = 'count',autopct='%1.1f%%',figsize=(8,8),legend=None,shadow=True, startangle=90)
plt.title('Probability of crops grow when rainfall > 150mm')
plt.show()
#which crops can grow at higher ph value .i.e., (alkaline nature) ph > 7.5
x = pd.DataFrame(pd.crosstab(data.label[data.ph > 7.5],'count',normalize=True)*100)
```

```
x.plot.pie(y = 'count',autopct='%1.1f%%',figsize=(8,8),legend=None,shadow=True, startangle=90)
plt.title('Probability of crops grow when ph > 7.5 i.e.,alkaline nature')
plt.show()
#Splitting the data into input and output
x = data.iloc[:,:-2]
y = data.Encoded_label
print('Input variables \n',x.head())
print('\nOutput Variable\n',y.head())
# Splitting the data into train and test
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)

print('Shape of Splitting:')
print('x_train = {}, x_test = {}, y_train = {}, y_test = {}'.format(x_train.shape, x_test.shape,
y_train.shape, y_test.shape))

# Importing necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
# Example usage after model training
model = LogisticRegression()
model.fit(x_train, y_train)

# Predict and display confusion matrix
y_pred = model.predict(x_test)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
# Assuming you have already trained the Logistic Regression model
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
# Generate predictions on the test set
pred_logis = model.predict(x_test)

# Classification report and accuracy score
print('REPORT : \n', classification_report(y_test, pred_logis))
acc_logis = accuracy_score(y_test, pred_logis)
print('Accuracy: ', acc_logis)

# Step 1: Import necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import pickle

# Step 2: Define the model and parameter grid
model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Step 3: Use GridSearchCV to find the best parameters
grid_rand = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')
grid_rand.fit(x_train, y_train)

# Step 4: Save the trained model to a pickle file
pickle_out = open('classifier.pkl', 'wb')
pickle.dump(grid_rand, pickle_out)
pickle_out.close()
```

```
print("Model saved as 'classifier.pkl'")

a = [[80,35,40,30,10000,-1,0]]

pickle_in = open('classifier.pkl','rb')

model = pickle.load(pickle_in)

pre = model.predict_proba(a)

pre = pd.DataFrame(data = np.round(pre.T*100,2),
index=classes.index,columns=['predicted_values'])

pre

high = pre.predicted_values.nlargest(5)

plt.figure(figsize=(15,10))

plt.rcParams['font.size']=15

plt.title('Crops Recommendations :',fontdict={'fontsize': 25, 'fontweight': 'medium'})

plt.pie(x=high,labels=high.index,autopct='%1.1f%%',explode=(0.1, 0, 0, 0,
0),shadow=True,startangle=90,

        colors=['green','red','cyan','brown','orange'])

plt.show()

# Assuming `pre.predicted_values` is a Pandas Series

highest = pre.predicted_values.nlargest(1)

# Iterate through the Series to extract the index (ind) and value (val)

for ind, val in highest.items():

    new_h = ind

# Output the highest prediction's index

new_h

# loading the dataset

crop_data=pd.read_csv('/content/crop_production.csv')

crop_data['Crop'] = crop_data['Crop'].str.lower()

#lst = list(crop_data['Crop'].str.lower().unique())

#lst.sort()

#print(lst)

crop_data['Crop'] = crop_data['Crop'].replace(['moth','peas  (vegetable)','bean','moong(green
gram)','pome granet','water
melon','cotton(lint)','gram'],['mothbeans','pigeonpeas','kidneybeans','mungbean','pomegranate','watermelon','cotton','chickpea'])

crop_data = crop_data[crop_data['Crop'].isin(['rice', 'maize', 'chickpea', 'kidneybeans',
'pigeonpeas','mothbeans', 'mungbean', 'blackgram', 'lentil', 'pomegranate','banana', 'mango',
'grapes', 'watermelon', 'apple','orange', 'papaya', 'coconut', 'cotton', 'jute', 'coffee'])]

crop_data = crop_data.drop(['State_Name','District_Name'],axis = 1)

crop_data

crop_data['Crop'].unique()

# dataset columns

crop_data.columns

crop_data.describe()

# Checking missing values of the dataset in each column

crop_data.isnull().sum()

# Dropping missing values

crop_data = crop_data.dropna()

crop_data

#checking

crop_data.isnull().values.any()

# Visualizing the features

ax = sns.pairplot(crop_data)

ax

data = crop_data

# Select only numeric columns

numeric_data = data.select_dtypes(include=['float64', 'int64'])

# Compute correlation

correlation_matrix = numeric_data.corr()

print(correlation_matrix)

dummy = pd.get_dummies(data)

dummy

from sklearn.model_selection import train_test_split

x = dummy.drop(["Production"], axis=1)

y = dummy["Production"]
```

```
# Splitting data set - 25% test dataset and 75% train
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25, random_state=5)
print("x_train :",x_train.shape)
print("x_test :",x_test.shape)
print("y_train :",y_train.shape)
print("y_test :",y_test.shape)
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 11)
model.fit(x_train,y_train)
rf_predict = model.predict(x_test)
rf_predict
model.score(x_test,y_test)
from sklearn.metrics import r2_score
r1 = r2_score(y_test,rf_predict)
print("R2 score : ",r1)
# Calculating Adj. R2 score:
Adjr2_1 = 1 - (1-r1)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
print("Adj. R-Squared : {}".format(Adjr2_1))
ax = sns.distplot(y_test, hist = False, color = "r", label = "Actual value ")
sns.distplot(rf_predict, hist = False, color = "b", label = "Predicted Values", ax = ax)
plt.title('Random Forest Regression')
inp = [2012, 'Kharif', new_h, 100]
# Select a test row
test_row = x_test.head(1)
# Modify the test_row with inputs
test_row['Crop_Year'] = inp[0]
for i in test_row.columns[2:]:
    string = str(i)
    if inp[1] in string or inp[2] in string:
        test_row[i] = 1
    else:
        test_row[i] = 0

test_row['Area'] = inp[3]

# Predict production
production = model.predict(test_row)[0]

# Calculate yield
yd = production / test_row['Area']
print("Production: ", production)

# Iterate through the Series using .items()
for ind, val in yd.items():
    print("Yield: ", val)
```