In [1]:
```
# importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# loading the dataset

crop_data=pd.read_csv("C:\\Users\\Admin\\Downloads\\cpdata.csv")
crop_data
```

Out[2]:

	temperature	humidity	ph	rainfall	label
0	20.879744	82.002744	6.502985	202.935536	rice
1	21.770462	80.319644	7.038096	226.655537	rice
2	23.004459	82.320763	7.840207	263.964248	rice
3	26.491096	80.158363	6.980401	242.864034	rice
4	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...
3095	25.287846	89.636679	6.765095	58.286977	watermelon
3096	26.638386	84.695469	6.189214	48.324286	watermelon
3097	25.331045	84.305338	6.904242	41.532187	watermelon
3098	26.897502	83.892415	6.463271	43.971937	watermelon
3099	26.986037	89.413849	6.260839	58.548767	watermelon

3100 rows × 5 columns

In [3]:
```
crop_data.shape

#rows X columns
```

Out[3]:
```
(3100, 5)
```

In [4]:
```
crop_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3100 entries, 0 to 3099
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   temperature  3100 non-null   float64
 1   humidity     3100 non-null   float64
 2   ph           3100 non-null   float64
 3   rainfall     3100 non-null   float64
 4   label        3100 non-null   object 
dtypes: float64(4), object(1)
memory usage: 121.2+ KB
```

In [5]:
```
# dataset columns
crop_data.columns
```

Out[5]:
```
Index(['temperature', 'humidity', 'ph', 'rainfall', 'label'], dtype='object')
```

In [6]:
```
crop_data.rename(columns = {'label':'Crop'}, inplace = True)
crop_data
```

Out[6]:

	temperature	humidity	ph	rainfall	Crop
0	20.879744	82.002744	6.502985	202.935536	rice
1	21.770462	80.319644	7.038096	226.655537	rice
2	23.004459	82.320763	7.840207	263.964248	rice
3	26.491096	80.158363	6.980401	242.864034	rice
4	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...
3095	25.287846	89.636679	6.765095	58.286977	watermelon
3096	26.638386	84.695469	6.189214	48.324286	watermelon
3097	25.331045	84.305338	6.904242	41.532187	watermelon
3098	26.897502	83.892415	6.463271	43.971937	watermelon
3099	26.986037	89.413849	6.260839	58.548767	watermelon

3100 rows × 5 columns

In [7]:
```
# statistical inference of the dataset

crop_data.describe()
```

Out[7]:

	temperature	humidity	ph	rainfall
count	3100.000000	3100.000000	3100.000000	3100.000000
mean	27.108466	66.005312	6.368913	110.213031
std	7.566308	24.007713	0.809477	64.048562
min	8.825675	10.034048	3.504752	20.211267
25%	22.810495	55.244920	5.895343	64.909095
50%	26.102848	68.980529	6.342518	97.057093
75%	29.365644	84.446524	6.841616	141.210784
max	54.986760	99.981876	9.935091	397.315380

In [8]:
```
# Checking missing values of the dataset in each column
crop_data.isnull().sum()
```

Out[8]:
```
temperature    0
humidity       0
ph             0
rainfall       0
Crop           0
dtype: int64
```

In [9]:
```
# Dropping missing values
crop_data = crop_data.dropna()
crop_data
```

Out[9]:

	temperature	humidity	ph	rainfall	Crop
0	20.879744	82.002744	6.502985	202.935536	rice
1	21.770462	80.319644	7.038096	226.655537	rice
2	23.004459	82.320763	7.840207	263.964248	rice
3	26.491096	80.158363	6.980401	242.864034	rice
4	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...
3095	25.287846	89.636679	6.765095	58.286977	watermelon
3096	26.638386	84.695469	6.189214	48.324286	watermelon
3097	25.331045	84.305338	6.904242	41.532187	watermelon
3098	26.897502	83.892415	6.463271	43.971937	watermelon
3099	26.986037	89.413849	6.260839	58.548767	watermelon

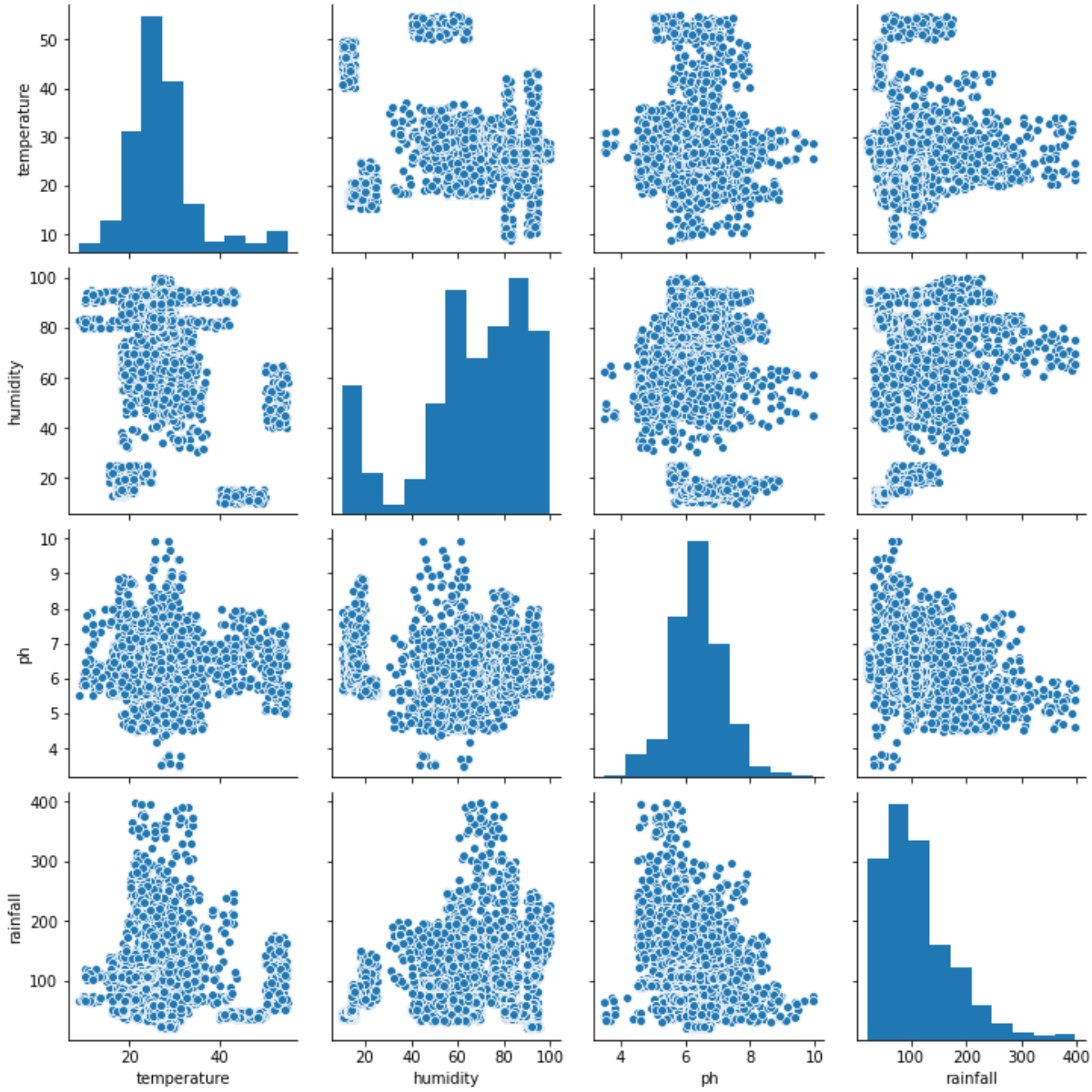
3100 rows × 5 columns

In [10]:
```
#checking
crop_data.isnull().values.any()
```

Out[10]:
```
False
```

In [11]:
```
# Visualizing the features

ax = sns.pairplot(crop_data)
ax
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x1c83c942f10>
```



In [12]:
```
crop_data.Crop.unique()
```

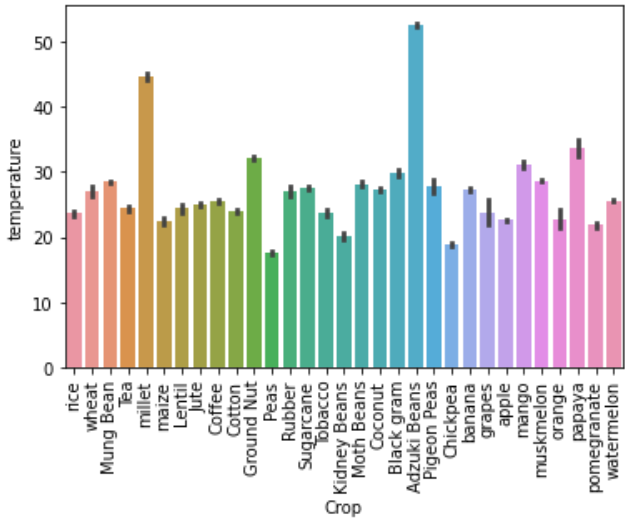
Out[12]:
```
array(['rice', 'wheat', 'Mung Bean', 'Tea', 'millet', 'maize', 'Lentil',
       'Jute', 'Coffee', 'Cotton', 'Ground Nut', 'Peas', 'Rubber',
       'Sugarcane', 'Tobacco', 'Kidney Beans', 'Moth Beans', 'Coconut',
       'Black gram', 'Adzuki Beans', 'Pigeon Peas', 'Chickpea', 'banana',
       'grapes', 'apple', 'mango', 'muskmelon', 'orange', 'papaya',
       'pomegranate', 'watermelon'], dtype=object)
```

In [13]:
```
# get top 5 most frequent growing crops
n = 5
crop_data['Crop'].value_counts()[:5].index.tolist()
```

Out[13]:
```
['Adzuki Beans', 'Mung Bean', 'Coffee', 'banana', 'orange']
```

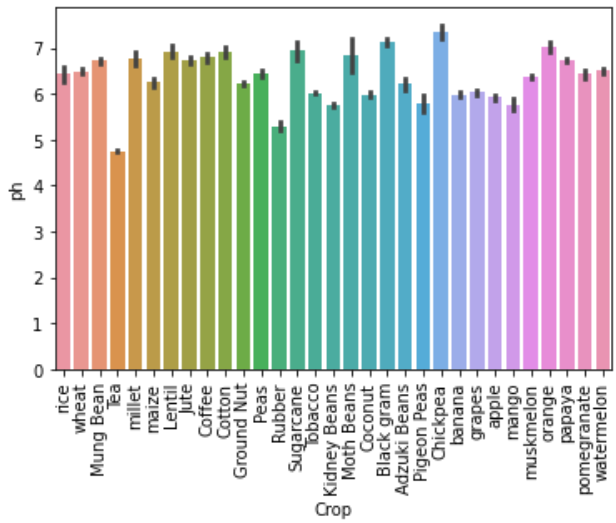
In [14]:
```
sns.barplot(crop_data["Crop"], crop_data["temperature"])
plt.xticks(rotation = 90)
```

Out[14]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]),
 <a list of 31 Text major ticklabel objects>)
```



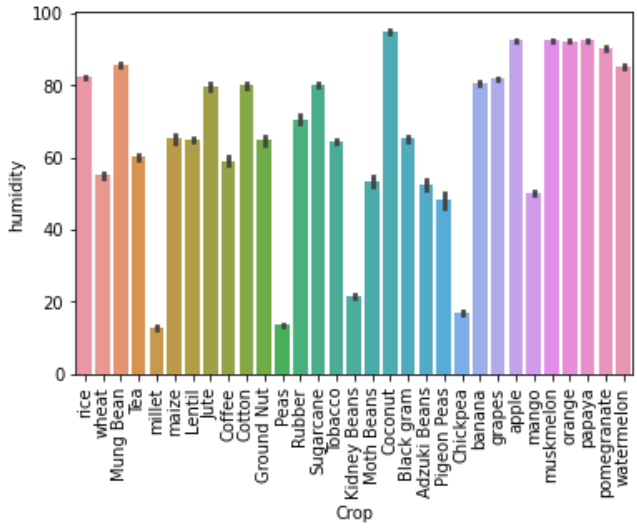
In [15]:
```
sns.barplot(crop_data["Crop"], crop_data["ph"])
plt.xticks(rotation = 90)
```

Out[15]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]),
 <a list of 31 Text major ticklabel objects>)
```



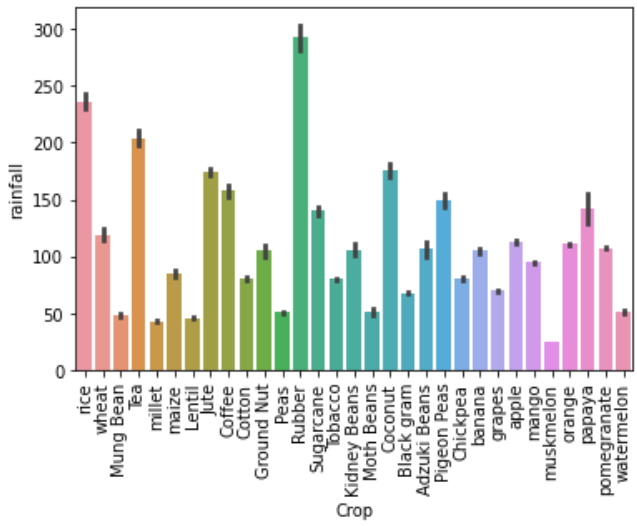
In [16]:
```
sns.barplot(crop_data["Crop"], crop_data["humidity"])
plt.xticks(rotation = 90)
```

Out[16]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]),
 <a list of 31 Text major ticklabel objects>)
```



In [17]:
```
sns.barplot(crop_data["Crop"], crop_data["rainfall"])
plt.xticks(rotation = 90)
```

Out[17]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]),
 <a list of 31 Text major ticklabel objects>)
```



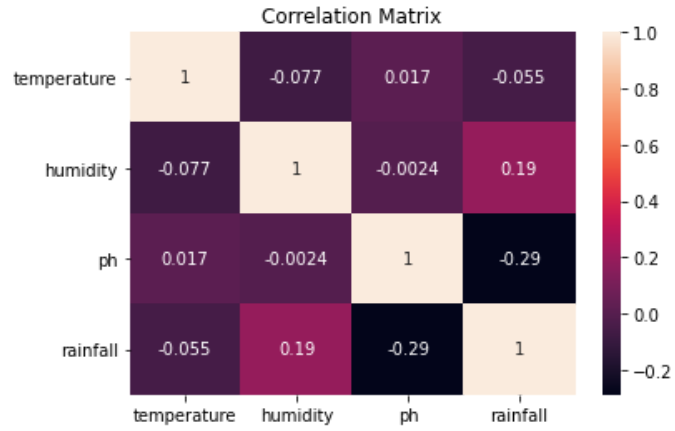
In [18]:
```
crop_data.corr()
```

Out[18]:

	temperature	humidity	ph	rainfall
temperature	1.000000	-0.076999	0.017024	-0.055143
humidity	-0.076999	1.000000	-0.002359	0.192074
ph	0.017024	-0.002359	1.000000	-0.288598
rainfall	-0.055143	0.192074	-0.288598	1.000000

In [19]:
```
sns.heatmap(crop_data.corr(), annot =True)
plt.title('Correlation Matrix')
```

Out[19]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```



In [20]:
```
# Shuffling data to remove order effects

# shuffling the dataset to remove order
from sklearn.utils import shuffle

df  = shuffle(crop_data,random_state=5)
df.head()
```

Out[20]:

	temperature	humidity	ph	rainfall	Crop
1141	16.912919	13.881680	5.959978	54.026676	Peas
2262	27.486130	76.112398	6.212369	109.276885	banana
1964	53.751483	61.805135	5.410117	130.090866	Adzuki Beans
1456	19.978657	63.458462	5.944788	84.685380	Tobacco
142	28.721646	59.375796	6.743792	121.484053	wheat

In [21]:
```
# Selection of Feature and Target variables.

x = df[['temperature', 'humidity', 'ph', 'rainfall']]
target = df['Crop']
```

In [22]:
```
# Encoding target variable

y = pd.get_dummies(target)
y
```

Out[22]:

	Adzuki Beans	Black gram	Chickpea	Coconut	Coffee	Cotton	Ground Nut	Jute	Kidney Beans	Lentil	...	maize	mango	millet	muskmelon	orange
1141	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2262	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1964	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1456	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
142	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1424	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3046	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1725	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0
2254	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2915	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

3100 rows × 31 columns

In [23]:
```
# Splitting data set - 25% test dataset and 75%


from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25, random_state= 0)

print("x_train :",x_train.shape)
print("x_test :",x_test.shape)
print("y_train :",y_train.shape)
print("y_test :",y_test.shape)
```

```
x_train : (2325, 4)
x_test : (775, 4)
y_train : (2325, 31)
y_test : (775, 31)
```

In [24]:
```
# Importing necessary libraries for multi-output classification

from sklearn.datasets import make_classification
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [25]:
```
# Training

forest = RandomForestClassifier(random_state=1)
multi_target_forest = MultiOutputClassifier(forest, n_jobs=-1)
multi_target_forest.fit(x_train, y_train)
```

Out[25]:
```
MultiOutputClassifier(estimator=RandomForestClassifier(random_state=1),
                      n_jobs=-1)
```

In [26]:
```
# Predicting test results

forest_pred = multi_target_forest.predict(x_test)
forest_pred
```

Out[26]:
```
array([[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, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [27]:
```
# Calculating Accuracy

from sklearn.metrics import accuracy_score
a1 = accuracy_score(y_test, forest_pred)
print('Accuracy score:', accuracy_score(y_test, forest_pred))
```

```
Accuracy score: 0.8529032258064516
```

Cross-validation

In [28]:
```
from sklearn.model_selection import cross_val_score
score = cross_val_score(multi_target_forest,X = x_train, y = y_train,cv=5)
score
```

Out[28]:
```
array([0.84946237, 0.8516129 , 0.88602151, 0.8344086 , 0.85376344])
```

In [29]:
```
b1 = "{:.2f}".format(score.mean()*100)
b1 = float(b1)
b1
```

Out[29]:
```
85.51
```

In [30]:
```
c1 = (score.std()*100)
c1
```

Out[30]:
```
1.6911486667817204
```

In [31]:
```
print("Accuracy : {:.2f}%".format (score.mean()*100))
print("Standard Deviation : {:.2f}%".format(score.std()*100))
```

```
Accuracy : 85.51%
Standard Deviation : 1.69%
```

In [32]:
```
# Training
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=6)
multi_target_decision = MultiOutputClassifier(clf, n_jobs=-1)
multi_target_decision.fit(x_train, y_train)
```

Out[32]:
```
MultiOutputClassifier(estimator=DecisionTreeClassifier(random_state=6),
                      n_jobs=-1)
```

In [33]:
```
# Predicting test results

decision_pred = multi_target_decision.predict(x_test)
decision_pred
```

Out[33]:
```
array([[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, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [34]:
```
# Calculating Accuracy

from sklearn.metrics import accuracy_score
a2 = accuracy_score(y_test,decision_pred)
print('Accuracy score:', accuracy_score(y_test,decision_pred))
a2
```

```
Accuracy score: 0.8309677419354838
```

Out[34]:
```
0.8309677419354838
```

Cross-validation

In [35]:
```
from sklearn.model_selection import cross_val_score
score = cross_val_score(multi_target_decision,X = x_train, y = y_train,cv=7)
score
```

Out[35]:
```
array([0.82882883, 0.86144578, 0.79216867, 0.88855422, 0.80722892,
       0.8253012 , 0.81024096])
```

In [36]:
```
b2 = "{:.2f}".format(score.mean()*100)
b2 = float(b2)
b2
```

Out[36]:
```
83.05
```

In [37]:
```
c2 = (score.std()*100)
c2
```

Out[37]:
```
3.1119598738808025
```

In [38]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_clf=KNeighborsClassifier()
model = MultiOutputClassifier(knn_clf, n_jobs=-1)
model.fit(x_train, y_train)
```

Out[38]:
```
MultiOutputClassifier(estimator=KNeighborsClassifier(), n_jobs=-1)
```

In [39]:
```
knn_pred = model.predict(x_test)
knn_pred
```

Out[39]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [40]:
```
# Calculating Accuracy

from sklearn.metrics import accuracy_score
a3 = accuracy_score(y_test,knn_pred)
print('Accuracy score:', accuracy_score(y_test,knn_pred))
a3
```

```
Accuracy score: 0.7974193548387096
```

Out[40]:
```
0.7974193548387096
```

Cross-validation

In [41]:
```
from sklearn.model_selection import cross_val_score
score = cross_val_score(model,X = x_train, y = y_train,cv=7)
score
```

Out[41]:
```
array([0.79279279, 0.80421687, 0.78313253, 0.78915663, 0.81325301,
       0.79819277, 0.77108434])
```

In [42]:
```
b3 = "{:.2f}".format(score.mean()*100)
b3 = float(b3)
b3
```

Out[42]:
```
79.31
```

In [43]:
```
c3 = (score.std()*100)
c3
```

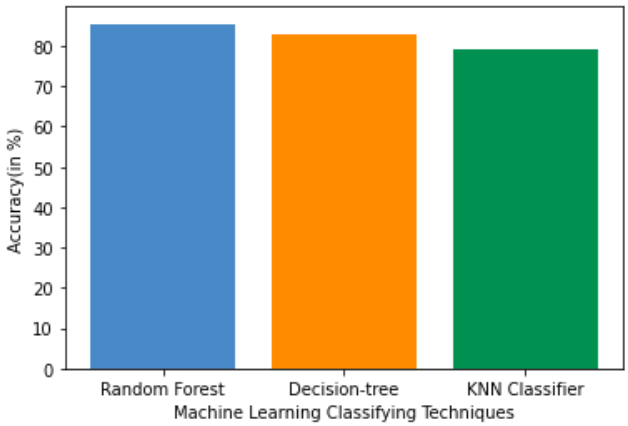
Out[43]:
```
1.2847162042492162
```

In [44]:
```
import pandas as pd

# initialise data of lists.
data = {'Algorithms':['Random Forest', 'Decision-tree', 'KNN Classifier'],
        'Accuracy':[b1, b2, b3],
        'Standard Deviation':[c1,c2,c3]}

# Creates pandas DataFrame.
df = pd.DataFrame(data)

# print the data
df
```

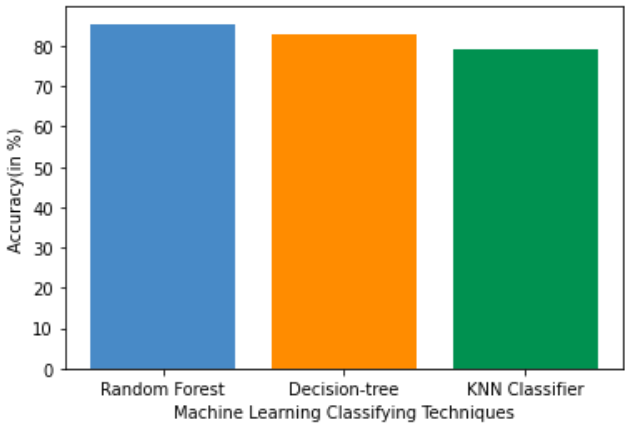
Out[44]:

	Algorithms	Accuracy	Standard Deviation
0	Random Forest	85.51	1.691149
1	Decision-tree	83.05	3.111960
2	KNN Classifier	79.31	1.284716

In [45]:
```
import numpy as np
import matplotlib.pyplot as plt

# create a dataset
Algorithms = ['Random Forest', 'Decision-tree','KNN Classifier']
Accuracy = [b1, b2, b3]

x_pos = np.arange(len(Accuracy))

# Create bars with different colors
plt.bar(x_pos, Accuracy, color=['#488AC7','#ff8c00','#009150'])

# Create names on the x-axis
plt.xticks(x_pos, Algorithms)
plt.ylabel('Accuracy(in %)')
plt.xlabel('Machine Learning Classifying Techniques')

# Show graph
plt.show()
```

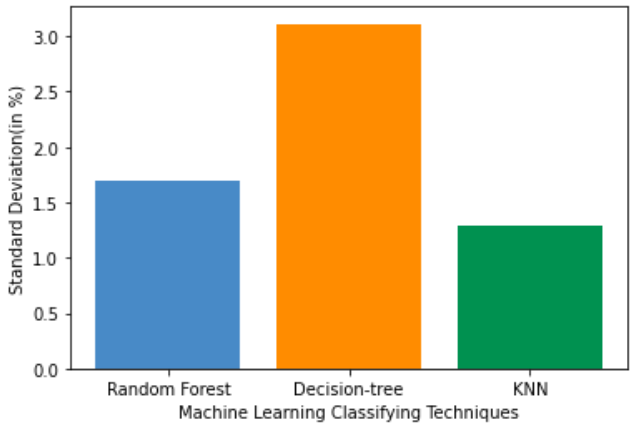


In [46]:
```
import numpy as np
import matplotlib.pyplot as plt

# create a dataset
Algorithms = ['Random Forest', 'Decision-tree','KNN Classifier']
Accuracy = [b1, b2, b3]

x_pos = np.arange(len(Accuracy))

# Create bars with different colors
plt.bar(x_pos, Accuracy, color=['#488AC7','#ff8c00','#009150'])

# Create names on the x-axis
plt.xticks(x_pos, Algorithms)
plt.ylabel('Accuracy(in %)')
plt.xlabel('Machine Learning Classifying Techniques')

# Show graph
plt.show()
```

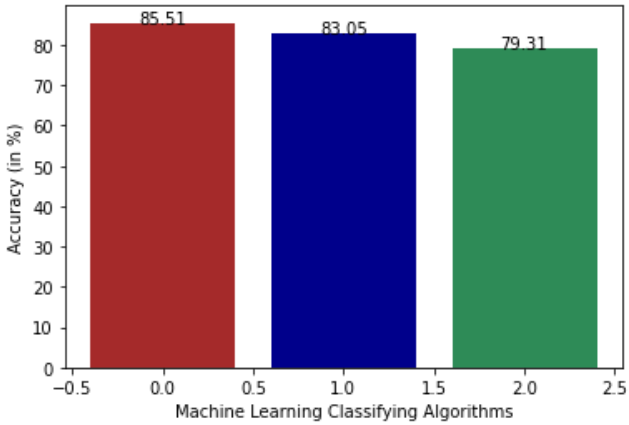


In [47]:

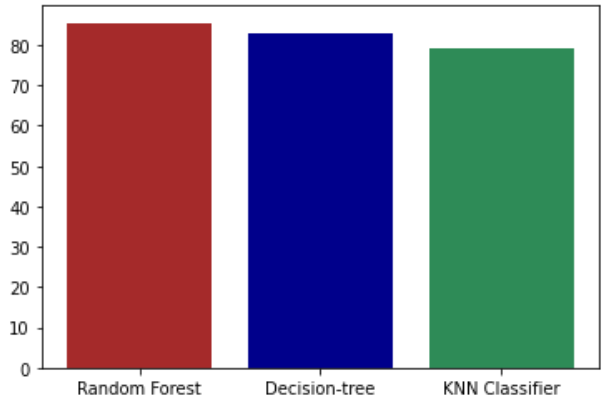
```
import numpy as np
import matplotlib.pyplot as plt

# create a dataset
Algorithms = ['Random Forest', 'Decision-tree','KNN']
Accuracy = [c1, c2, c3]

x_pos = np.arange(len(Accuracy))

# Create bars with different colors
plt.bar(x_pos, Accuracy, color= ['#488AC7','#ff8c00','#009150'])

# Create names on the x-axis
plt.xticks(x_pos, Algorithms)
plt.ylabel('Standard Deviation(in %)')
plt.xlabel('Machine Learning Classifying Techniques')

# Show graph
plt.show()
```

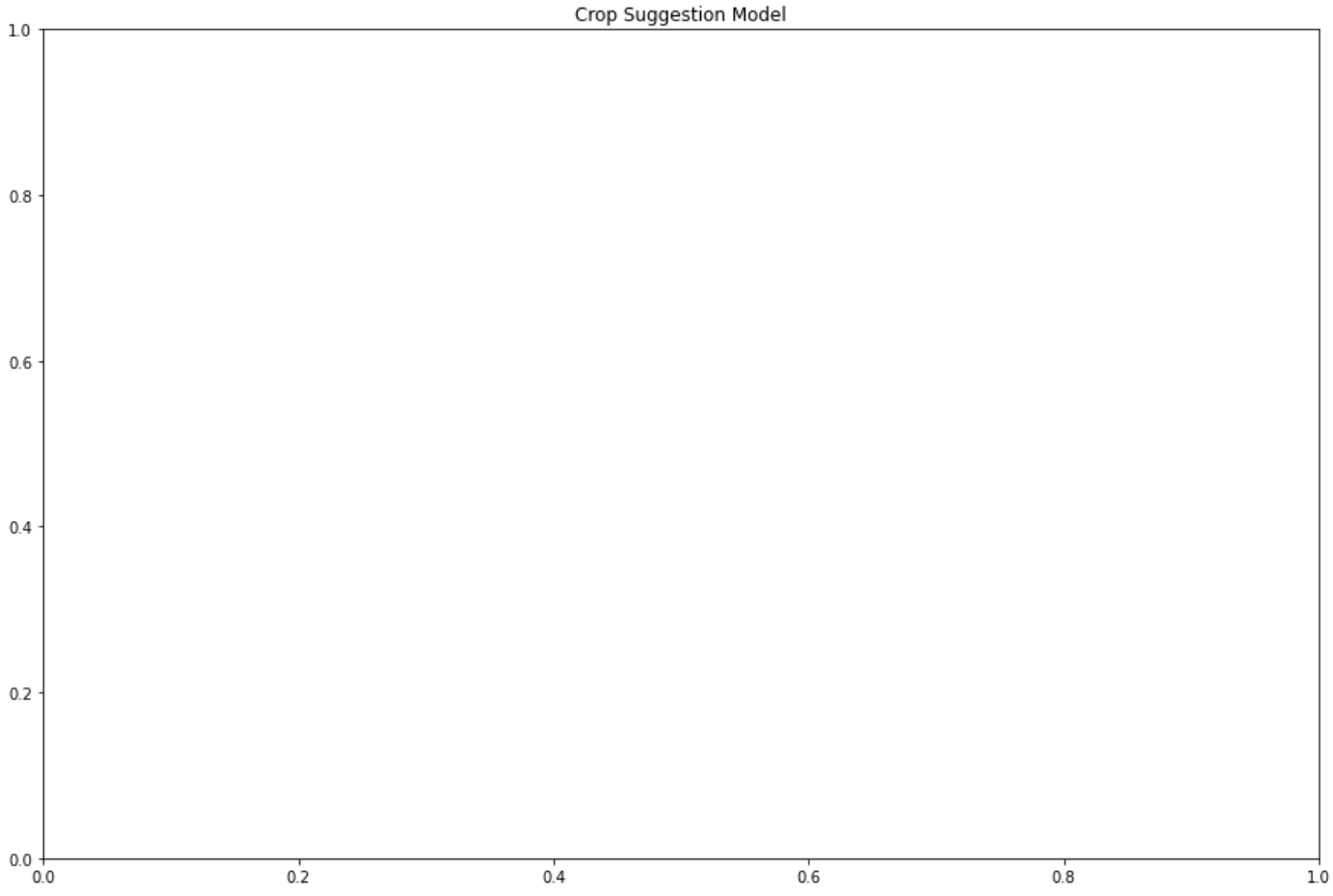


In [48]:
```
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i],ha = 'center')

if __name__ == '__main__':
    # creating data on which bar chart will be plot
    x = ["Random Forest", "Decision tree", "KNN"]
    y = [b1,b2,b3]


    x_pos = np.arange(len(y))

    # Create bars with different colors
    plt.bar(x_pos, y, color= ['#A52A2A','#00008B','#2E8B57'])

    # calling the function to add value labels
    addlabels(x, y)

    # giving X and Y labels
    plt.xlabel("Machine Learning Classifying Algorithms")
    plt.ylabel("Accuracy (in %)")

    # visualizing the plot
    plt.show()
```

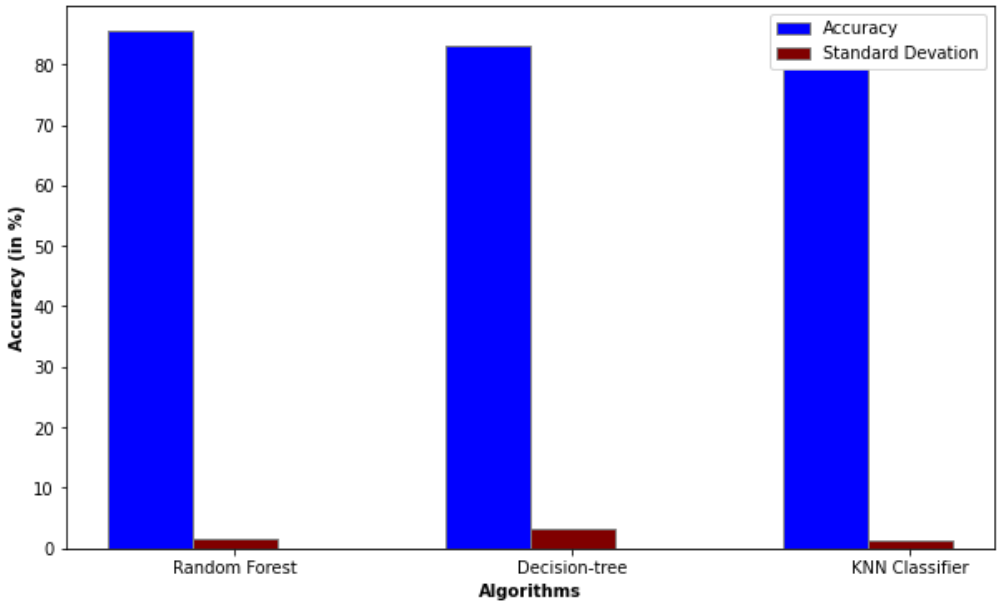


In [49]:
```
plt.bar(df['Algorithms'], df['Accuracy'], color = ['#A52A2A','#00008B','#2E8B57'])
fig = plt.figure(figsize =(15, 10))
plt.title('Crop Suggestion Model')

# Show Plot
plt.show()
```





In [50]:
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(10, 6))

# set height of bar
Algorithms = ['Random Forest', 'Decision-tree', 'KNN Classifier']
Accuracy = [b1, b2, b3]
Standard_Deviation = [c1,c2,c3]

# Set position of bar on X axis
br1 = np.arange(len(Accuracy))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

# Make the plot
plt.bar(br1, Accuracy, color ='blue', width = barWidth,
        edgecolor ='grey', label ='Accuracy')
plt.bar(br2, Standard_Deviation, color ='maroon', width = barWidth,
        edgecolor ='grey', label ='Standard Devation')

# Adding Xticks
plt.xlabel('Algorithms', fontweight ='bold', fontsize = 10)
plt.ylabel('Accuracy (in %)', fontweight ='bold', fontsize = 10)
plt.xticks([r + barWidth for r in range(len(Accuracy))],
        Algorithms)

plt.legend()
plt.show()
```



In [51]:
```
# Saving the trained Random Forest model
import pickle
# Dump the trained Naive Bayes classifier with Pickle
RF_pkl_filename = 'RandomForest.pkl'
# Open the file to save as pkl file
RF_Model_pkl = open(RF_pkl_filename, 'wb')
pickle.dump(multi_target_forest, RF_Model_pkl)
# Close the pickle instances
RF_Model_pkl.close()
```