ABOUT THE DATASET:

- The data refers to district wise, crop wise, season wise and year wise data on crop covered area (Hectare) and production (Tonnes).
- The data is being used to study and analyse crop production, production contribution to district/State/country, Agro-climatic zone wise performance, and high yield production order for crops, crop growing pattern and diversification.
- The system is also a vital source for formulating crop related schemes and assessing their impacts.

Code, documentation, and explanations.

• Data exploration and pre-processing:

1. Data cleaning
2. Visualization
3. Pre-processing.

• Data cleaning

In [1]:
```
# importing the required  libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Explore the data to understand its structure, identify the variables, and check for any missing values.

In [2]:
```
#importing the data
data = pd.read_csv("C:/Users/chanu/Downloads/apy.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	State_Name	District_Name	Crop_Year	Season	Crop	Area	Production
0	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Arecanut	1254.0	2000.0
1	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Other Kharif pulses	2.0	1.0
2	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Rice	102.0	321.0
3	Andaman and Nicobar Islands	NICOBARS	2000	Whole Year	Banana	176.0	641.0
4	Andaman and Nicobar Islands	NICOBARS	2000	Whole Year	Cashewnut	720.0	165.0

In [4]:
```
#dimensions of the data
data.shape
```

Out[4]: (246091, 7)

The essential libraries, such as NumPy, pandas, matplotlib, and seaborn, are imported first in the code sample above. The dataset is a csv file called apy. As data, a data frame is built using the pandas library. To begin exploring the data, we printed the first few rows and determined how many rows and columns were there in the data.

In [5]:
```
#print the columns/features of the data
data.columns
```

Out[5]:
```
Index(['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area',
       'Production'],
      dtype='object')
```

In [6]:
```
# listing the numericale features
num_cols=data.select_dtypes(include=np.number).columns
num_cols
```

Out[6]:
```
Index(['Crop_Year', 'Area', 'Production'], dtype='object')
```

In [7]:
```
# listing the categorical features
cat_cols=data.select_dtypes(include='object').columns
cat_cols
```

Out[7]:
```
Index(['State_Name', 'District_Name', 'Season', 'Crop'], dtype='object')
```

In [8]:
```
data.describe()
```

Out[8]:

	Crop_Year	Area	Production
count	246091.000000	2.460910e+05	2.423610e+05
mean	2005.643018	1.200282e+04	5.825034e+05
std	4.952164	5.052340e+04	1.706581e+07
min	1997.000000	4.000000e-02	0.000000e+00
25%	2002.000000	8.000000e+01	8.800000e+01
50%	2006.000000	5.820000e+02	7.290000e+02
75%	2010.000000	4.392000e+03	7.023000e+03
max	2015.000000	8.580100e+06	1.250800e+09

Printing the names of the columns, separating them into numerical and category columns, and then explaining the various statistical information such as mean, mode, standard deviation, min, max, quartiles.

Handle missing values by imputing the missing values with mean

In [9]:
```
# checking for missing values
data.null=data.isnull().sum()
data.null
```

```
C:\Users\chanu\AppData\Local\Temp\ipykernel_16720\1504268037.py:2: UserWarning: Pandas doesn't allow columns to be created via
a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  data.null=data.isnull().sum()
```

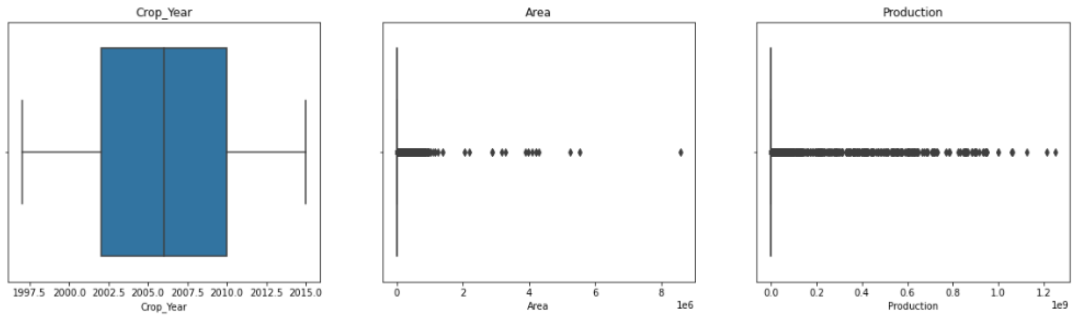
Out[9]:
```
State_Name          0
District_Name       0
Crop_Year           0
Season              0
Crop                0
Area                0
Production       3730
dtype: int64
```

In [10]:
```
# # filling the null values with mean value of column having null values
mean_value=data['Production'].mean()
data['Production']=data['Production'].fillna(mean_value)
```

The data frame may contain null values in each column. So, after determining which and all columns contain null values, the null values are filled using the mean of the associated column. Only the production column in our data frame has 3730 null entries. Filling those with the production column's mean.

Handle outliers:Outliers are extreme values that can skew the analysis and lead to inaccurate results. Capping involves setting a threshold for the maximum and minimum values of a variable and replacing any values outside this range with the threshold value.

In [11]:
```python
import matplotlib.pyplot as plt

# Assuming 'data' is your pandas DataFrame
numeric_cols = data.select_dtypes(include=[np.number]).columns.tolist()

# Create subplots for each numeric column
fig, axs = plt.subplots(ncols=len(numeric_cols), figsize=(20, 5))

# Iterate over each numeric column and plot the outliers using boxplot
for i, col in enumerate(numeric_cols):
    sns.boxplot(x=data[col], ax=axs[i])
    axs[i].set_title(col)
```



In [12]:
```python
def cap_data(data):
    for col in data.columns:
        print("\n\n capping the \n",col)
        if (((data[col].dtype)=='float64') | ((data[col].dtype)=='int64')):

            q1=data[col].quantile(0.25)
            q3=data[col].quantile(0.75)
            iqr=q3-q1
            lower,upper=(q1-(iqr*1.5)),(q3+(iqr*1.5))
            print("q1=",q1,"q3=",q3,"iqr=",iqr,"lower=",lower,"upper=",upper)
            data[col][data[col] <= lower] = lower
            data[col][data[col] >= upper] = upper
            print("\n",data[col][data[col] <= lower] )
            print("\n",data[col][data[col] >= upper] )

        else:
            data[col]=data[col]
    return data

final_df=cap_data(data)
```

```
 capping the 
 State_Name


 capping the 
 District_Name


 capping the 
 Crop_Year
q1= 2002.0 q3= 2010.0 iqr= 8.0 lower= 1990.0 upper= 2022.0

 Series([], Name: Crop_Year, dtype: int64)

 Series([], Name: Crop_Year, dtype: int64)


 capping the 
 Season


 capping the 
 Crop


 capping the 
 Area
q1= 80.0 q3= 4392.0 iqr= 4312.0 lower= -6388.0 upper= 10860.0

 Series([], Name: Area, dtype: float64)

 5          10860.0
14         10860.0
23         10860.0
32         10860.0
41         10860.0
            ...   
246017    10860.0
246033    10860.0
246052    10860.0
246070    10860.0
246089    10860.0
Name: Area, Length: 40707, dtype: float64


 capping the 
 Production
q1= 91.0 q3= 8000.0 iqr= 7909.0 lower= -11772.5 upper= 19863.5

 Series([], Name: Production, dtype: float64)

 5          19863.5
14         19863.5
23         19863.5
32         19863.5
41         19863.5
            ...   
245985    19863.5
246017    19863.5
246043    19863.5
246052    19863.5
246089    19863.5
Name: Production, Length: 43787, dtype: float64
```

Using a box plot, we can see which rows and columns include outliers. Because we have numerical columns for area production and crop_year, we used a boxplot to determine that only the production column has outliers.

The capping method is being used to remove outliers. First, we discover the 1st and 3rd quartile values, as well as the iqr value, and then we get the low and high values based on these values. If the datapoints are less than low, they are allocated a low value; similarly, if the datapoints are greater than high, they are assigned a high value.

Handle duplicates: Remove any duplicate data points to avoid any bias in the analysis.

In [13]:
```
# identify duplicate rows based on all columns
duplicates = data[data.duplicated()]

# keep the first instance of each duplicate row
data.drop_duplicates(keep='first', inplace=True)

# print the updated DataFrame
print(data)
```

```
                         State_Name District_Name  Crop_Year       Season  \
0       Andaman and Nicobar Islands      NICOBARS       2000  Kharif
1       Andaman and Nicobar Islands      NICOBARS       2000  Kharif
2       Andaman and Nicobar Islands      NICOBARS       2000  Kharif
3       Andaman and Nicobar Islands      NICOBARS       2000  Whole Year
4       Andaman and Nicobar Islands      NICOBARS       2000  Whole Year
...                             ...           ...        ...          ...
246086                  West Bengal       PURULIA       2014  Summer
246087                  West Bengal       PURULIA       2014  Summer
246088                  West Bengal       PURULIA       2014  Whole Year
246089                  West Bengal       PURULIA       2014  Winter
246090                  West Bengal       PURULIA       2014  Winter

                     Crop     Area  Production
0                Arecanut   1254.0      2000.0
1        Other Kharif pulses    2.0         1.0
2                    Rice    102.0       321.0
3                  Banana    176.0       641.0
4               Cashewnut    720.0       165.0
...                   ...      ...         ...
246086               Rice    306.0       801.0
246087            Sesamum    627.0       463.0
246088          Sugarcane    324.0     16250.0
246089               Rice  10860.0     19863.5
246090            Sesamum    175.0        88.0

[246091 rows x 7 columns]
```

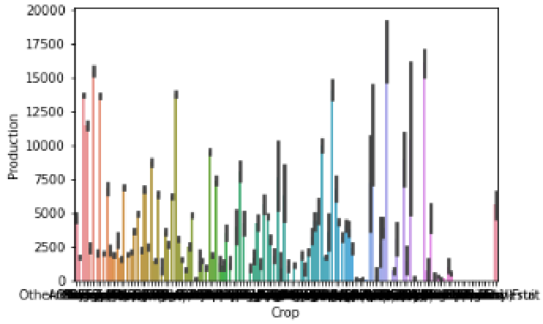
If there are duplicate values in the data frame, we must eliminate them. Identify the duplicates first, then delete all values except the first instance of that row in the data frame. We don't have any duplicates in our data frame because the row number remains the same.

• Visualization

Crop-wise production contribution: bar chart can be used to show the contribution of different crops to the total production in each district or state. The x-axis can represent the districts or states and the y-axis can represent the production in tonnes. Each crop can be represented by a different color within the bar chart.we can use hue for displaying the production for each based on diffrent crop.

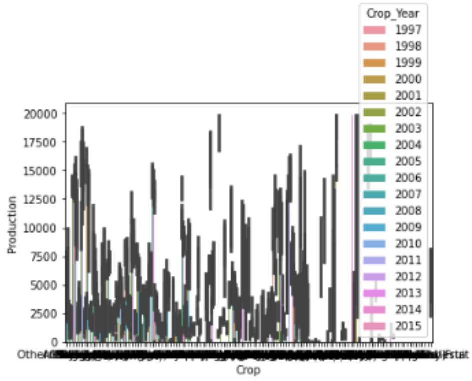
In [14]:
```
sns.barplot(x='Crop',y='Production',data=data)
```

Out[14]: <AxesSubplot:xlabel='Crop', ylabel='Production'>



In [15]:
```python
# Adding one more parameter in the graph by using HUE
sns.barplot(x='Crop',y='Production',hue='Crop_Year',data=data)
```

Out[15]: <AxesSubplot:xlabel='Crop', ylabel='Production'>



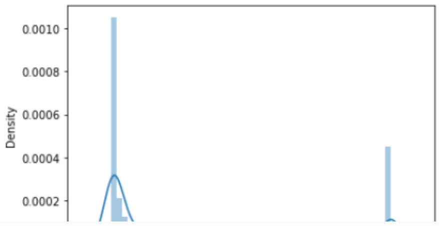
Density charts are used to examine a variable's distribution in a dataset. Because production is the variable in this case, the distribution on the graph is right skewed.

Density Plot for production

In [16]:
```python
sns.distplot(data['Production'])
```

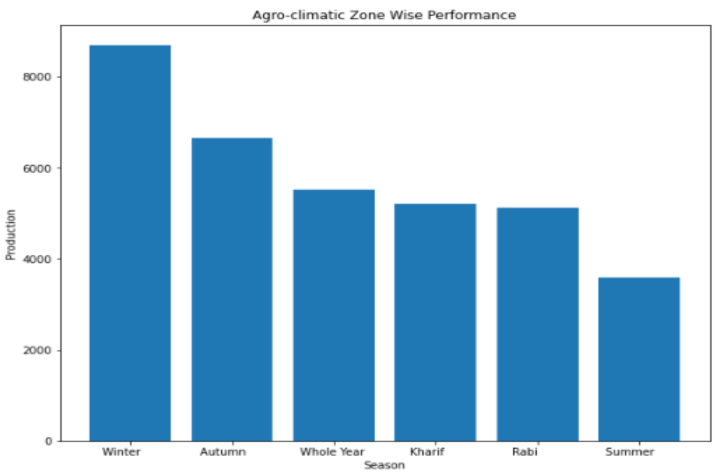
```
C:\Users\chanu\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[16]: <AxesSubplot:xlabel='Production', ylabel='Density'>



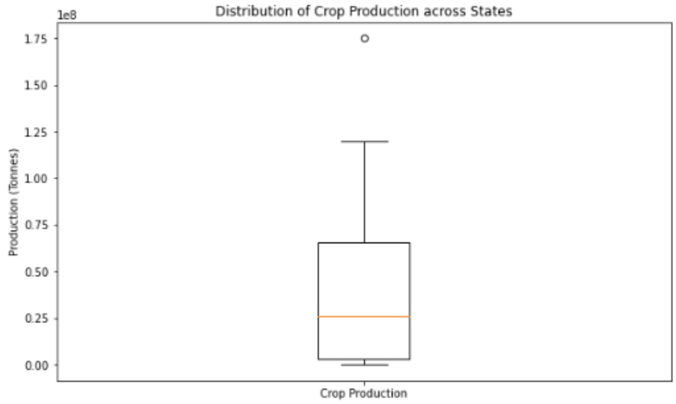
Agro-climatic zone wise performance: A bar chart can be used to show the agro-climatic zone wise performance, where each bar represents a different zone and the height of each bar represents the performance score.

In [18]:
```python
# Group the data by zone and compute the average performance score
grouped_data = data.groupby(['Season'])['Production'].mean().reset_index()

# Sort the data by performance score in descending order
sorted_data = grouped_data.sort_values(by=['Production'], ascending=False)

# Create a bar chart
plt.figure(figsize=(10, 8))
plt.bar(sorted_data['Season'], sorted_data['Production'])
plt.xlabel('Season')
plt.ylabel('Production')
plt.title('Agro-climatic Zone Wise Performance')
plt.show()
```



Formulating crop-related schemes and assessing their impacts: A box plot can be used to show the distribution of crop production across different states. The x-axis can represent the states and the y-axis can represent the production in tonnes.The box plot can show the minimum, maximum, median, and quartiles of production, providing insights into the variability of production across different states.

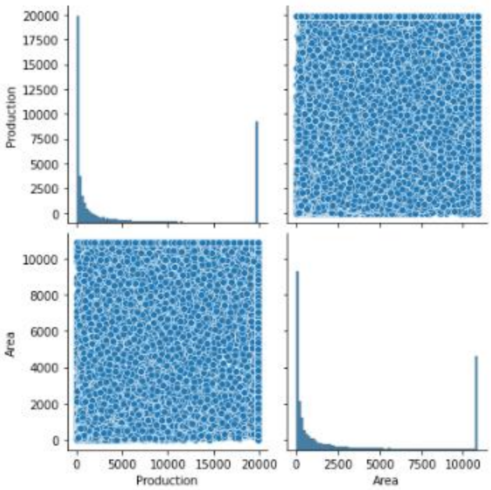
In [19]:
```
# Group the data by district/state and crop, and compute the total production
grouped_data = data.groupby(['State_Name'])['Production'].sum().reset_index()

# Plot a box plot
plt.figure(figsize=(10, 6))
plt.boxplot(grouped_data['Production'])
plt.xticks([1], ['Crop Production'])
plt.title('Distribution of Crop Production across States')
plt.ylabel('Production (Tonnes)')
plt.show()
```



Piar plot is used to plot multiple pairwise bivariate distributionin a dataset.

In [20]:
```
features=['Production','Area']
sns.pairplot(data[features],height=3)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x27db7767a90>



Visualizing a dataset can be very helpful for gaining insights into the data and identifying patterns and trends that may not be immediately apparent from the raw data. Some common uses of visualizing a dataset include:

Exploratory data analysis: Data visualization can help in exploring the data to understand its structure, patterns, and relationships.

Feature selection: Visualizing the dataset can help in selecting the most important features that have the strongest correlation with the target variable.

Outlier detection: Data visualization can help in identifying any unusual or unexpected data points that may be outliers and need to be treated separately.

Data pre-processing: Visualizing the dataset can help in identifying any missing data, outliers, or inconsistencies in the data, which can be cleaned or imputed before training a machine learning model.

•Pre-processing.

Encoding is used to convert categorical data into a numerical format that can be used by machine learning algorithms. This can include technique label encoding.

In [21]:
```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
le = LabelEncoder()

# Define the columns to encode
columns_to_encode = ['State_Name','District_Name','Season','Crop']

# Loop through each column and encode the categories
for col in columns_to_encode:
    encoded_col = le.fit_transform(data[col])
    data[col] = encoded_col
```

In [22]:
```
data
```

Out[22]:

	State_Name	District_Name	Crop_Year	Season	Crop	Area	Production
0	0	427	2000	1	2	1254.0	2000.0
1	0	427	2000	1	74	2.0	1.0
2	0	427	2000	1	95	102.0	321.0
3	0	427	2000	4	7	176.0	641.0
4	0	427	2000	4	22	720.0	165.0
...	...	...	...	...	...	...	...
246086	32	471	2014	3	95	306.0	801.0
246087	32	471	2014	3	102	627.0	463.0
246088	32	471	2014	4	106	324.0	16250.0
246089	32	471	2014	5	95	10860.0	19863.5
246090	32	471	2014	5	102	175.0	88.0

246091 rows × 7 columns

The categorical columns in our data set are state_name, district_name, season, and crop. As a result, we must turn them into numerical columns using the label coding function, for which we must import the sklearn pre-processing package.

•Feature engineering

constructing a heatmap to understand the correlation between the columns

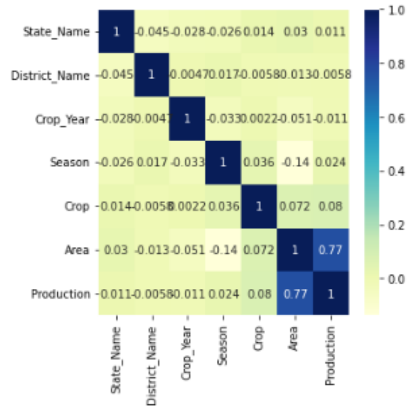
Sampling is used to reduce the size of a dataset, while preserving its key characteristics. This can be useful because we are working with large datasets that are difficult to process.

In [25]:
```
df= data.astype(int)
sampled_df = df.sample(n=246090)
sampled_df.head()
```

Out[25]:

	State_Name	District_Name	Crop_Year	Season	Crop	Area	Production
15658	3	152	2005	2	57	25	12
193473	28	366	2007	1	95	10860	19863
158573	22	492	2001	1	59	3746	3070
167334	25	90	2000	4	32	6	12
89996	14	230	2003	1	48	10860	10977

```
Crop_Year     -0.051246   -0.010804
Season        -0.138270    0.024494
Crop           0.071707    0.080425
Area           1.000000    0.767170
Production     0.767170    1.000000
```



Using the heat map, determine the correlation between the columns with each other; if any columns are poorly associated, eliminate them; otherwise, keep them. There is a strong correlation between all of the columns here.

Because the data frame comprises 246090 rows, we can create the model using a sampling strategy. We can adjust the size dependent on the model's accuracy.

The columns serve as features and labels. So, in this case, x stands for features (state_name, district_name, season, crop, crop_year, area) and y stands for the label (Production). Printing the few rows of the data frame x and y. Then they are divided into training and testing data. The test and train data sizes can be any size; in this case, the ratio is 80:20. ravel is a function that converts a 2-dimensional or multi-dimensional array into a contiguous-flattened-array.

splitting the columns as features as x and label as y

In [26]:
```
from sklearn.model_selection import train_test_split
x = sampled_df[['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area']]
# y= data['Production']
y=sampled_df .drop(['State_Name', 'District_Name', 'Crop_Year', 'Season', 'Crop', 'Area'],axis='columns')
```

In [27]:
```
x.head(3)
```

Out[27]:

	State_Name	District_Name	Crop_Year	Season	Crop	Area
15658	3	152	2005	2	57	25
193473	28	366	2007	1	95	10860
158573	22	492	2001	1	59	3746

In [28]:
```
y.head(3)
```

Out[28]:

	Production
15658	12
193473	19863
158573	3070

Split the dataset as training data and testing data

In [41]:
```
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=42)
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()
```

• Model selection: Choose an appropriate algorithm for the problem and justify your choice.

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Define a list of model names and their corresponding hyperparameters
models = [
    ('Linear Regression', LinearRegression()),
    ('Ridge Regression', Ridge(alpha=0.1)),
    ('Lasso Regression', Lasso(alpha=0.1)),
    ('Decision Tree Regression', DecisionTreeRegressor(max_depth=5,min_samples_split=5)),
    ('Random Forest Regression', RandomForestRegressor(n_estimators=100, random_state=42))
]

# Define an empty dictionary to store the performance metrics for each model
metrics = {}

# Iterate over each model and train and evaluate it on the training data
for name, model in models:
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    mse = mean_squared_error(y_test, y_pred)
    metrics[name] = mse

# Print the performance metrics for each model
for name, mse in metrics.items():
    print(name, 'MSE:', mse)

# Select the best model based on the performance metrics
best_model_name = min(metrics, key=metrics.get)
best_model = next(model for model in models if model[0] == best_model_name)[1]
print('Best model:', best_model_name)
```

```
Linear Regression MSE: 22583162.41598582
Ridge Regression MSE: 22583162.409309775
Lasso Regression MSE: 22583161.36538914
Decision Tree Regression MSE: 19334863.298575178
Random Forest Regression MSE: 4837343.041668302
Best model: Random Forest Regression
```

The choice of a suitable algorithm is critical in model construction. Linear regression, ridge regression, lasso regression, decision tree regression, and random forest regression are some of the algorithms available for regression problems. So we run a loop to see which algorithm is best with some parameters for the algorithm to return the best value. For regression, we utilise mean squared

•Model optimization: Optimize the model by tuning its hyper-parameters and/or using regularization techniques.
•Model training and evaluation:Train the model on the dataset and evaluate its performance using appropriate metrics.

In [43]:
```
from sklearn.model_selection import GridSearchCV
model = RandomForestRegressor()
# hyperparameters
parameters = {
              'n_estimators': [50, 100,],
              'random_state': [32,42]
}
```

In [46]:
```
# grid search
classifier = GridSearchCV(model, parameters, cv=5)
# fitting the data to our model
model=classifier.fit(x_train, y_train)
```

In [56]:
```
classifier.cv_results_
```

Out[56]:
```
{'mean_fit_time': array([24.05237865, 22.90589561, 46.54866934, 46.23114042]),
 'std_fit_time': array([0.72375267, 0.26227719, 0.53675601, 0.20242086]),
 'mean_score_time': array([0.70168815, 0.68566461, 1.33396211, 1.35008483]),
 'std_score_time': array([0.03838334, 0.02344351, 0.03644026, 0.03262101]),
 'param_n_estimators': masked_array(data=[50, 50, 100, 100],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_random_state': masked_array(data=[32, 42, 32, 42],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 50, 'random_state': 32},
  {'n_estimators': 50, 'random_state': 42},
  {'n_estimators': 100, 'random_state': 32},
  {'n_estimators': 100, 'random_state': 42}],
 'split0_test_score': array([0.9074854 , 0.90744654, 0.90850504, 0.90823142]),
 'split1_test_score': array([0.90851674, 0.90803847, 0.90956151, 0.90899831]),
 'split2_test_score': array([0.90643935, 0.90580079, 0.90742131, 0.90675816]),
 'split3_test_score': array([0.90617279, 0.90655263, 0.90792044, 0.90783219]),
 'split4_test_score': array([0.91043877, 0.91002273, 0.91114108, 0.91089832]),
 'mean_test_score': array([0.90781061, 0.90757223, 0.90890988, 0.90854368]),
 'std_test_score': array([0.00155399, 0.00144331, 0.00132375, 0.00138151]),
 'rank_test_score': array([3, 4, 1, 2])}
```

In [57]:
```
# best parameters

best_parameters = classifier.best_params_
print(best_parameters)
```

```
{'n_estimators': 100, 'random_state': 32}
```

error to evaluate the algorithms, and the approach with the lowest mean squared error is chosen to build the model.so the random forest regression algorithm is best as it has less mse value.
The process of determining the best settings for a machine learning model's hyperparameters is known as tuning. Hyperparameters are settings or configurations that are selected prior to model training, such as n_estimators, random_state, and so on.
Grid Search is one of the strategies for hyperparameter tweaking. This technique defines a set of hyperparameters and generates a grid of all possible combinations. For each combination, the model is trained and assessed, and the combination with the best performance is picked. As a result, the optimum parameters for random forest regression are n_estimators=100 and random_state=32.

```
In [58]: # higest accuracy
         highest_accuracy = classifier.best_score_
         print(highest_accuracy)

         0.9089098764742287

In [59]: # Loading the results to pandas dataframe
         result = pd.DataFrame(classifier.cv_results_)

In [60]: result.head()

Out[60]:
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_random_state	params	split0_test_score	split1_test_score
0	24.052379	0.723753	0.701688	0.038383	50	32	{'n_estimators': 50, 'random_state': 32}	0.907485	0.908517
1	22.905896	0.262277	0.685665	0.023444	50	42	{'n_estimators': 50, 'random_state': 42}	0.907447	0.908038
2	46.548669	0.536756	1.333962	0.036440	100	32	{'n_estimators': 100, 'random_state': 32}	0.908505	0.909562
3	46.231140	0.202421	1.350085	0.032621	100	42	{'n_estimators': 100, 'random_state': 42}	0.908231	0.908998

Checking the best score for the model produced with selected hyper parameter technique parameters and printing the result for each combination of hyper parameter technique parameters.

```
In [63]: from sklearn.metrics import r2_score
         print("Mean absolute error: %.2f" % np.mean(np.absolute(y_pred - y_test)))
         print("Residual sum of squares (MSE): %.2f" % np.mean((y_pred - y_test) ** 2))
         print("R2-score: %.2f" % r2_score(y_test,y_pred ) )

         Mean absolute error: 856.35
         Residual sum of squares (MSE): 4837343.04
         R2-score: 0.92

In [64]: print('Variance score: %.2f' % model.score(x_test,y_test))

         Variance score: 0.92

In [65]: import pickle
         filename = 'CROP_PRODUCTION_prediction_pickle.sav'
         pickle.dump(model, open(filename, 'wb'))
         # Loading the saved model
         loaded_model = pickle.load(open('CROP_PRODUCTION_prediction_pickle.sav', 'rb'))
```

To evaluate the model's performance, calculate the mean absolute error, mean squared error, and r2_score.The model's score is then checked, and it is good, indicating that the model is performing properly.The model is saved for future usage while being deployed with the pickle library.