India Agriculture Crop Production Prediction

India is a large producer of several agricultural products. In terms of quantity of production,one of the world's largest and most diverse agricultural producers, making it an ideal domain for machine learning applications in agriculture. The country's agriculture sector is a significant contributor to its economy, employing a large portion of its workforce and providing food security for its population

AIM: Predict the India Agriculture Crop Production

ABOUT THE DATASET: Dataset from KAGGLE

IMPORTING LIBRARIES

In [1]:
```
import numpy as np      #numerical operations
import pandas as pd     #data manipulation
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('/content/India Agriculture.csv')
df
```

Out [1]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andaman and Nicobar Islands	NICOBARS	Arecanut	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	Andaman and Nicobar Islands	NICOBARS	Arecanut	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	Andaman and Nicobar Islands	NICOBARS	Arecanut	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	Andaman and Nicobar Islands	NORTH AND MIDDLE ANDAMAN	Arecanut	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	Andaman and Nicobar Islands	SOUTH ANDAMANS	Arecanut	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296
...	...	...	...	...	...	...	...	...	...	...
345402	Manipur	IMPHAL WEST	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	Manipur	SENAPATI	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	Manipur	TAMENGLONG	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	Manipur	THOUBAL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	Manipur	UKHRUL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

345407 rows × 10 columns

In [2]:
```
df.shape
```

Out [2]:
```
(345407, 10)
```

In [3]:
```
df.dtypes
```

Out [3]:
```
State                object
District             object
Crop                 object
Year                 object
Season               object
Area                float64
Area Units           object
Production          float64
Production Units     object
Yield               float64
dtype: object
```

In [4]:
```
df.head()
```

Out [4]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andaman and Nicobar Islands	NICOBARS	Arecanut	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	Andaman and Nicobar Islands	NICOBARS	Arecanut	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	Andaman and Nicobar Islands	NICOBARS	Arecanut	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	Andaman and Nicobar Islands	NORTH AND MIDDLE ANDAMAN	Arecanut	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	Andaman and Nicobar Islands	SOUTH ANDAMANS	Arecanut	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296

In [5]:
```
df.tail()
```

Out [5]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
345402	Manipur	IMPHAL WEST	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	Manipur	SENAPATI	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	Manipur	TAMENGLONG	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	Manipur	THOUBAL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	Manipur	UKHRUL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

In [6]:
```
df.describe()
```

Out [6]:

	Area	Production	Yield
count	3.453740e+05	3.404140e+05	345374.000000
mean	1.167019e+04	9.583711e+05	79.407569
std	4.583843e+04	2.152986e+07	916.628744
min	4.000000e-03	0.000000e+00	0.000000
25%	7.400000e+01	8.700000e+01	0.546742
50%	5.320000e+02	7.170000e+02	1.000000
75%	4.110000e+03	7.176000e+03	2.467080
max	8.580100e+06	1.597800e+09	43958.333330

In [7]:
```
df.isna().sum()
```

Out [7]:
```
State                  0
District               0
Crop                  32
Year                   0
Season                 1
Area                  33
Area Units             0
Production          4993
Production Units       0
Yield                 33
dtype: int64
```

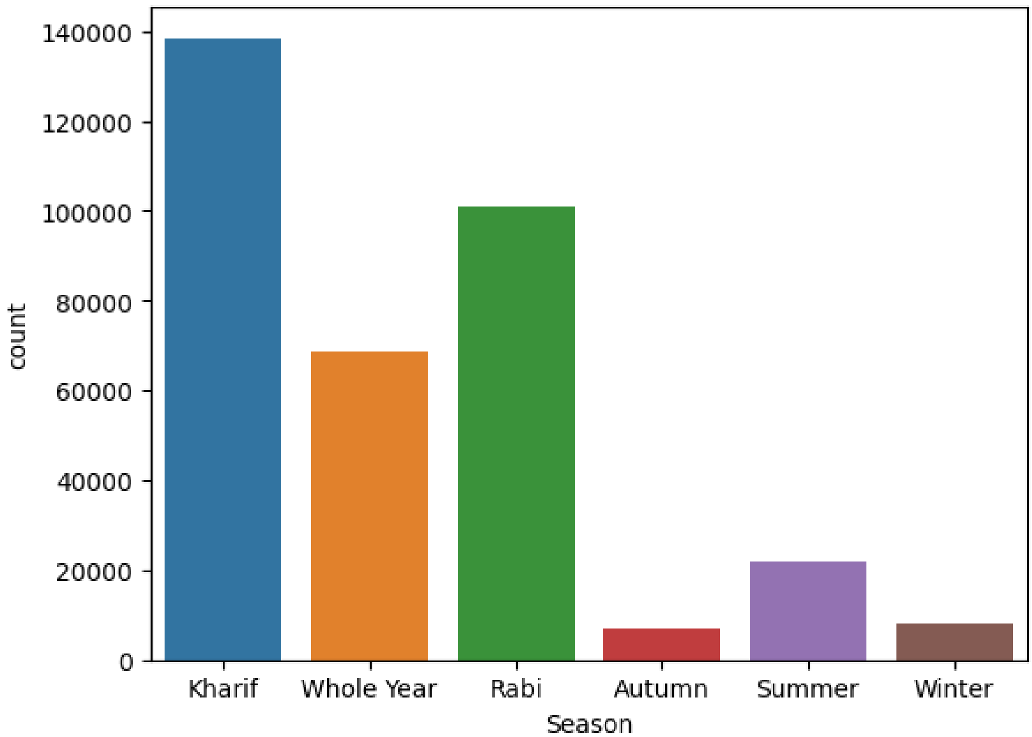
Exploratory data analysis

In [8]:
```
df=df.drop_duplicates()
df.shape
```

Out [8]:
```
(345407, 10)
```

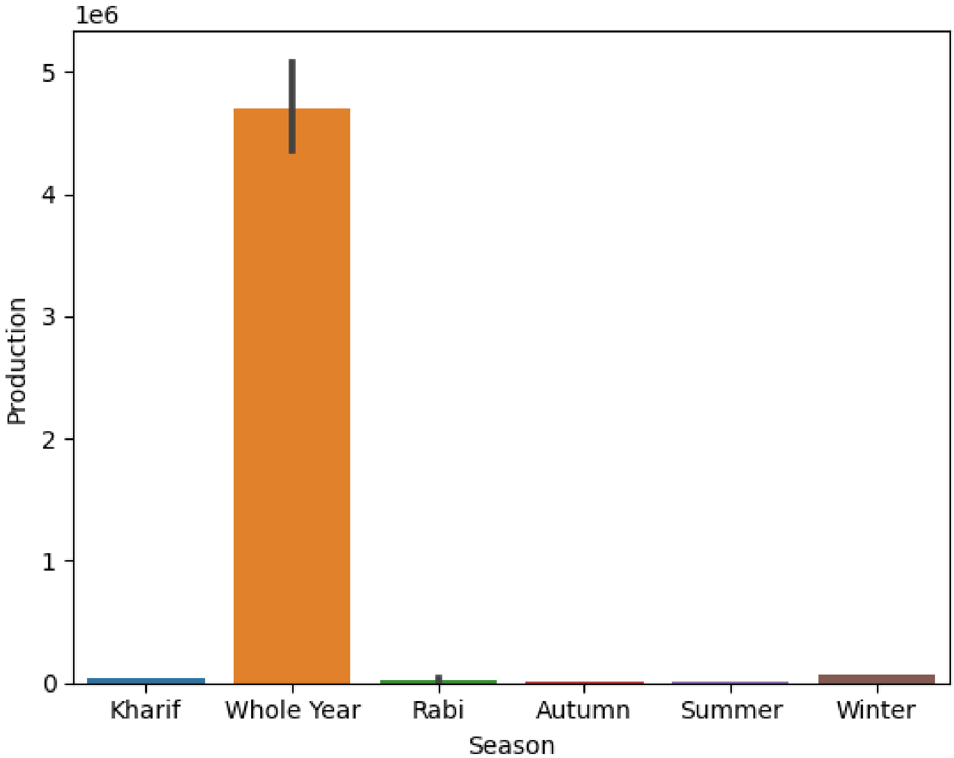
In [9]:
```
sns.countplot(x=df['Season'])
```

Out [9]:
```
<Axes: xlabel='Season', ylabel='count'>
```



In [10]:
```
sns.barplot(x=df['Season'],y=df['Production'])
```

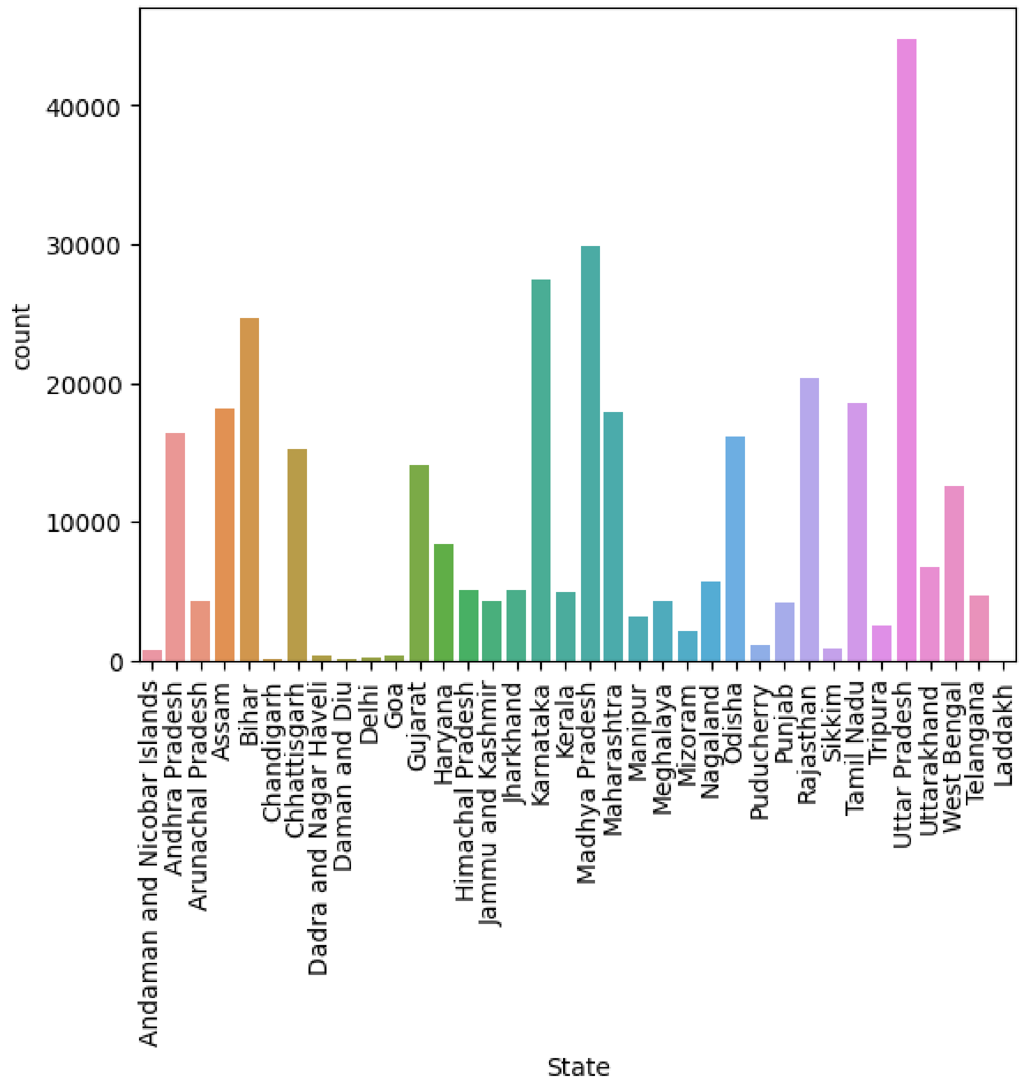
Out [10]: <Axes: xlabel='Season', ylabel='Production'>



In [11]:
```
sns.countplot(x='State',data=df)
plt.xticks(rotation=90)
```

Out [11]:
```
(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, 31, 32, 33,
```

```
       34, 35]),
[Text(0, 0, 'Andaman and Nicobar Islands'),
 Text(1, 0, 'Andhra Pradesh'),
 Text(2, 0, 'Arunachal Pradesh'),
 Text(3, 0, 'Assam'),
 Text(4, 0, 'Bihar'),
 Text(5, 0, 'Chandigarh'),
 Text(6, 0, 'Chhattisgarh'),
 Text(7, 0, 'Dadra and Nagar Haveli'),
 Text(8, 0, 'Daman and Diu'),
 Text(9, 0, 'Delhi'),
 Text(10, 0, 'Goa'),
 Text(11, 0, 'Gujarat'),
 Text(12, 0, 'Haryana'),
 Text(13, 0, 'Himachal Pradesh'),
 Text(14, 0, 'Jammu and Kashmir'),
 Text(15, 0, 'Jharkhand'),
 Text(16, 0, 'Karnataka'),
 Text(17, 0, 'Kerala'),
 Text(18, 0, 'Madhya Pradesh'),
 Text(19, 0, 'Maharashtra'),
 Text(20, 0, 'Manipur'),
 Text(21, 0, 'Meghalaya'),
 Text(22, 0, 'Mizoram'),
 Text(23, 0, 'Nagaland'),
 Text(24, 0, 'Odisha'),
 Text(25, 0, 'Puducherry'),
 Text(26, 0, 'Punjab'),
 Text(27, 0, 'Rajasthan'),
 Text(28, 0, 'Sikkim'),
 Text(29, 0, 'Tamil Nadu'),
 Text(30, 0, 'Tripura'),
 Text(31, 0, 'Uttar Pradesh'),
 Text(32, 0, 'Uttarakhand'),
 Text(33, 0, 'West Bengal'),
 Text(34, 0, 'Telangana'),
 Text(35, 0, 'Laddakh')])
```



In [12]:
```
#  Top 10 Crops
df["Crop"].value_counts().head(10)
```

Out [12]:
```
Rice                  21611
Maize                 20507
Moong(Green Gram)     15101
Urad                  14581
Sesamum               13049
Groundnut             12586
Wheat                 11248
Rapeseed &Mustard     11034
Sugarcane             10942
Arhar/Tur             10895
Name: Crop, dtype: int64
```

In [13]:
```
df=df.drop(['Area Units','Production Units'],axis=1)
```

In [14]:
```
df['State'].value_counts()
```

Out [14]:
```
Uttar Pradesh                  44781
Madhya Pradesh                 29906
Karnataka                      27493
Bihar                          24697
Rajasthan                      20363
Tamil Nadu                     18525
Assam                          18186
Maharashtra                    17922
Andhra Pradesh                 16363
Odisha                         16153
Chhattisgarh                   15285
Gujarat                        14053
West Bengal                    12596
Haryana                         8305
Uttarakhand                     6702
Nagaland                        5676
Himachal Pradesh                5043
Jharkhand                       5004
Kerala                          4870
Telangana                       4704
Jammu and Kashmir               4348
Arunachal Pradesh               4345
Meghalaya                       4322
Punjab                          4142
Manipur                         3120
Tripura                         2557
Mizoram                         2112
Puducherry                      1127
Sikkim                           876
Andaman and Nicobar Islands      728
Goa                              399
Dadra and Nagar Haveli           332
Delhi                            203
Chandigarh                       124
Daman and Diu                     44
Laddakh                            1
Name: State, dtype: int64
```

In [15]:
```
df['Year'].value_counts()
```

Out [15]:
```
2019-20    19296
2018-19    18302
2017-18    18008
2016-17    17418
2015-16    16339
2013-14    16178
2011-12    16132
2014-15    15587
2009-10    15341
2012-13    15279
2008-09    15150
2010-11    14889
2007-08    14681
2006-07    14678
2003-04    14662
2002-03    14182
2004-05    14151
2005-06    14063
2000-01    13593
2001-02    13307
1999-00    13013
1998-99    12290
1997-98     8549
2020-21      319
Name: Year, dtype: int64
```

In [16]:
```
df['Area'].value_counts()
```

Out [16]:
```
1.0         6638
2.0         4963
100.0       3877
3.0         3770
4.0         3277
            ...
46093.0        1
22233.0        1
52675.0        1
28912.0        1
52147.0        1
Name: Area, Length: 47713, dtype: int64
```

In [17]:
```
df['District'].value_counts()
```

Out [17]:
```
BILASPUR                                    1244
BIJAPUR                                     1218
AURANGABAD                                  1164
DAVANGERE                                   1151
HAVERI                                      1147
                                            ...
CHARAIDEO                                      2
BISWANATH                                      2
SOUTH SALMARA MANCACHAR                        1
THE DADRA AND NAGAR HAVELI AND DAMAN AND DIU    1
MUMBAI                                         1
Name: District, Length: 729, dtype: int64
```

In [18]:
```
#filling missing values
df['Production']=df['Production'].fillna((df['Production']).mean())
df['Crop']=df['Crop'].fillna((df['Crop']).mode()[0])
df['Area']=df['Area'].fillna((df['Area']).mean())
df['Yield']=df['Yield'].fillna((df['Yield']).mean())        #numerical===mean
df['Season']=df['Season'].fillna((df['Season']).mode()[0])  #object===mode
```

In [19]:
```
df.isna().sum()
```

Out [19]:
```
State         0
District      0
Crop          0
Year          0
Season        0
Area          0
Production    0
Yield         0
dtype: int64
```

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 345407 entries, 0 to 345406
Data columns (total 8 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   State       345407 non-null  object
 1   District    345407 non-null  object
 2   Crop        345407 non-null  object
 3   Year        345407 non-null  object
 4   Season      345407 non-null  object
 5   Area        345407 non-null  float64
 6   Production  345407 non-null  float64
 7   Yield       345407 non-null  float64
dtypes: float64(3), object(5)
memory usage: 23.7+ MB
```

DATA PREPROCESSING

In [21]:
```
#label encoding to the categorical label into numerical label
#each unique category is  assigned a unique integer
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
df['State']=le.fit_transform(df['State'])
df['District']=le.fit_transform(df['District'])
df['Crop']=le.fit_transform(df['Crop'])
df['Season']=le.fit_transform(df['Season'])
df['Year']=le.fit_transform(df['Year'])
```

In [22]:
```
df
```

Out [22]:

	State	District	Crop	Year	Season	Area	Production	Yield
0	0	481	0	4	1	1254.000000	2061.000000	1.643541
1	0	481	0	5	4	1258.000000	2083.000000	1.655803
2	0	481	0	6	4	1261.000000	1525.000000	1.209358
3	0	485	0	4	1	3100.000000	5239.000000	1.690000
4	0	627	0	5	4	3105.000000	5267.000000	1.696296
...	...	...	...	...	...	...	...	...
345402	21	269	41	22	2	11670.191258	958371.148664	79.407569
345403	21	586	41	22	2	11670.191258	958371.148664	79.407569
345404	21	647	41	22	2	11670.191258	958371.148664	79.407569
345405	21	662	41	22	2	11670.191258	958371.148664	79.407569
345406	21	683	41	22	2	11670.191258	958371.148664	79.407569

345407 rows × 8 columns

In [23]:
```
df.dtypes
```

Out [23]:
```
State          int64
District       int64
Crop           int64
Year           int64
Season         int64
Area         float64
Production   float64
Yield        float64
dtype: object
```

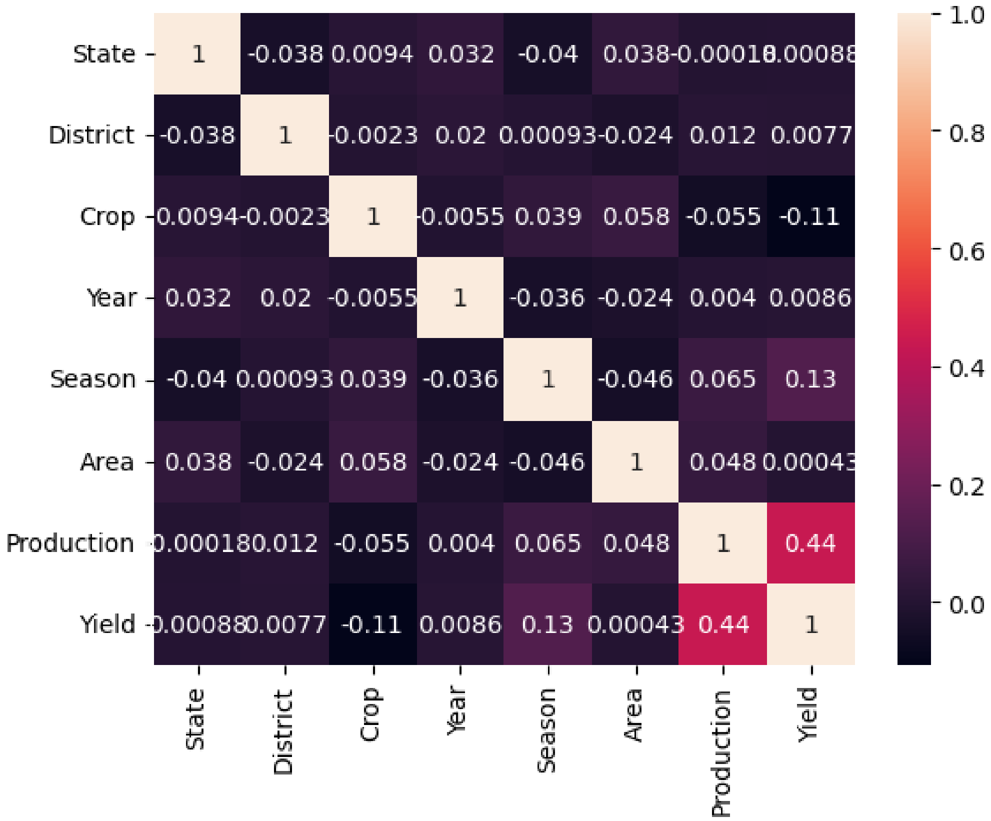
In [24]:
```
df.corr()  #pairwise correlation coeffient between the numeric columns
```

Out [24]:

	State	District	Crop	Year	Season	Area	Production	Yield
State	1.000000	-0.037775	0.009447	0.031537	-0.040313	0.037712	-0.000178	0.000882
District	-0.037775	1.000000	-0.002308	0.019817	0.000926	-0.023610	0.012218	0.007666
Crop	0.009447	-0.002308	1.000000	-0.005510	0.038615	0.058074	-0.054864	-0.107402
Year	0.031537	0.019817	-0.005510	1.000000	-0.036244	-0.024222	0.004029	0.008557
Season	-0.040313	0.000926	0.038615	-0.036244	1.000000	-0.045767	0.065277	0.128983
Area	0.037712	-0.023610	0.058074	-0.024222	-0.045767	1.000000	0.048472	0.000426
Production	-0.000178	0.012218	-0.054864	0.004029	0.065277	0.048472	1.000000	0.437376
Yield	0.000882	0.007666	-0.107402	0.008557	0.128983	0.000426	0.437376	1.000000

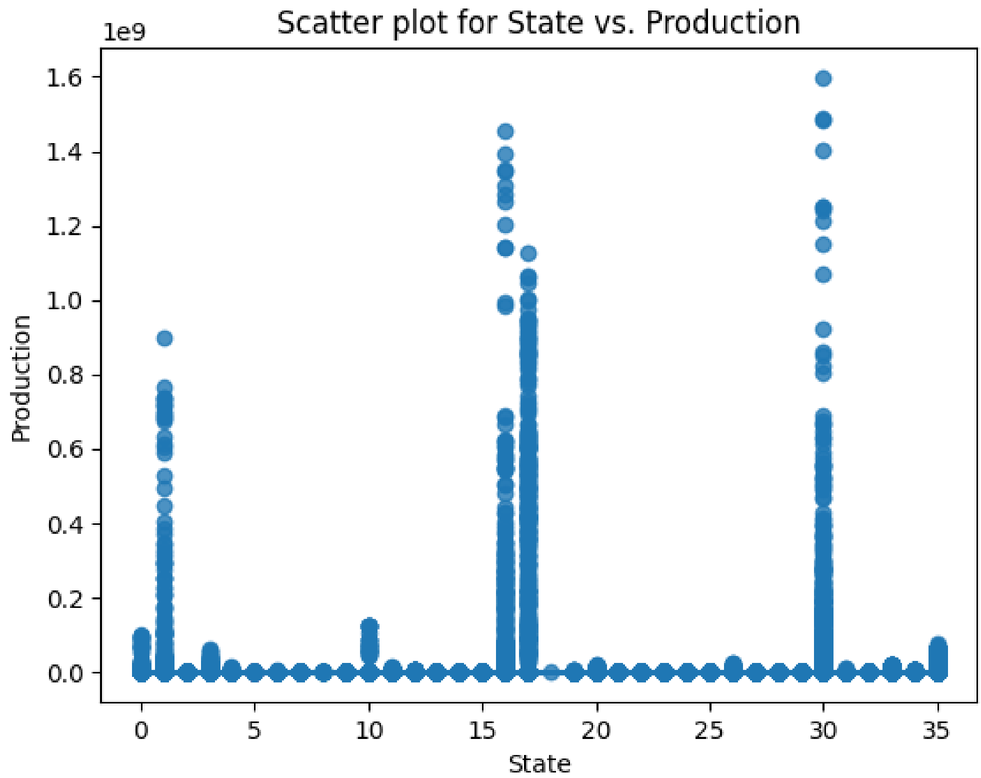
In [25]:
```
#correlation represented in graphical using Heatmap()
sns.heatmap(df.corr(),annot=True) #it displayes correlation matrix
```

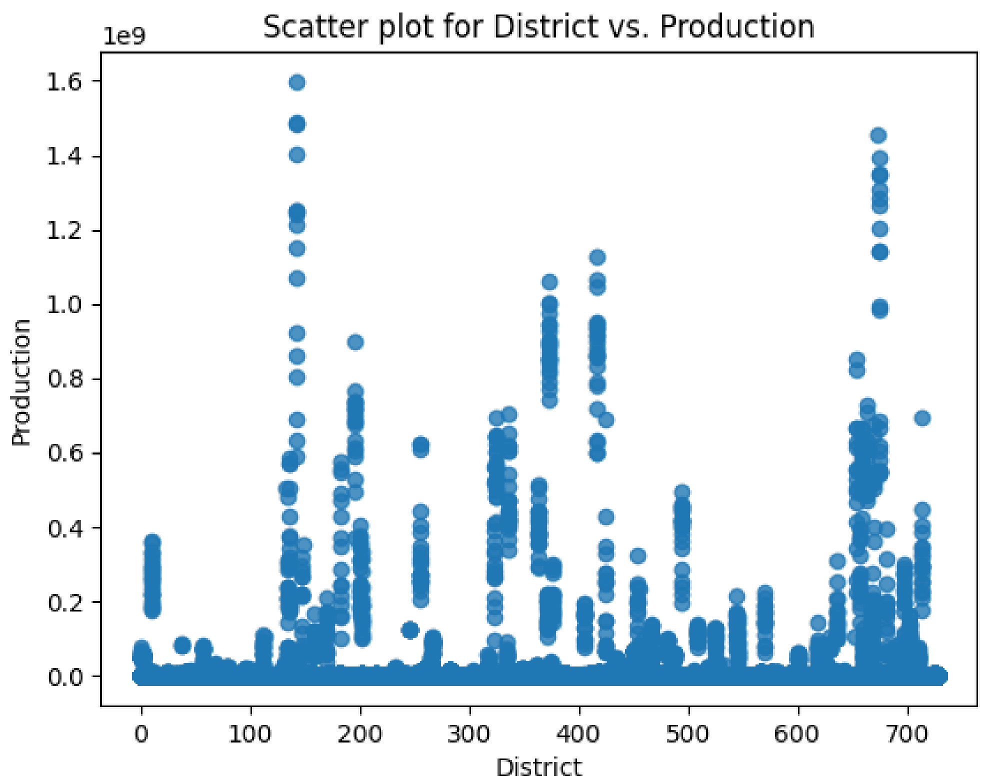
Out [25]: <Axes: >

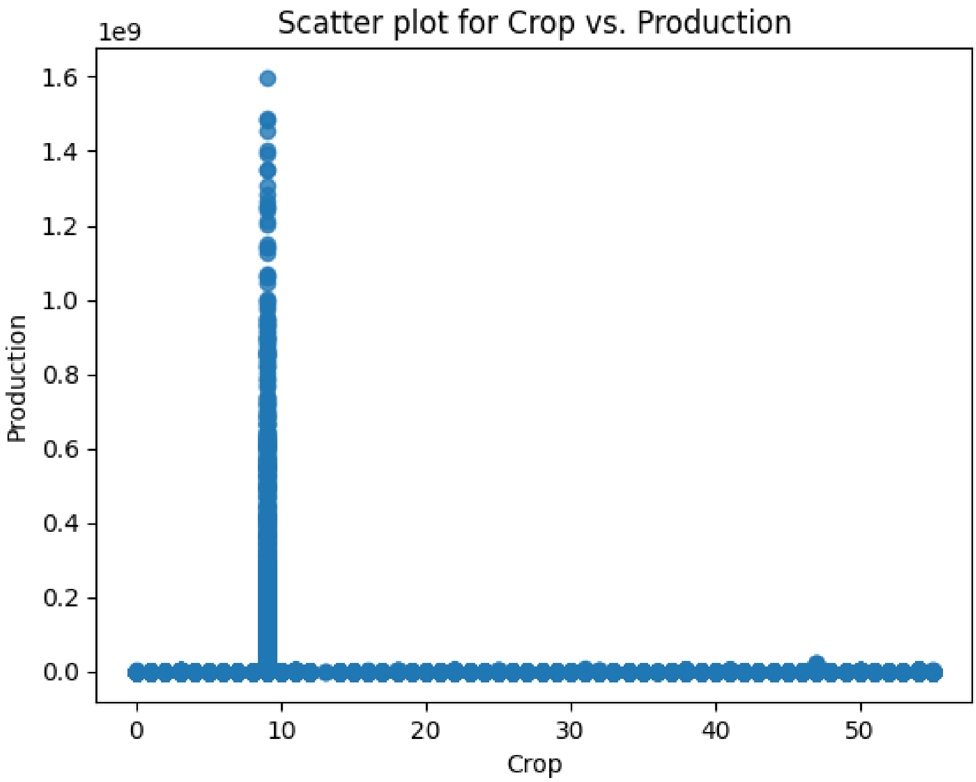


In [26]:

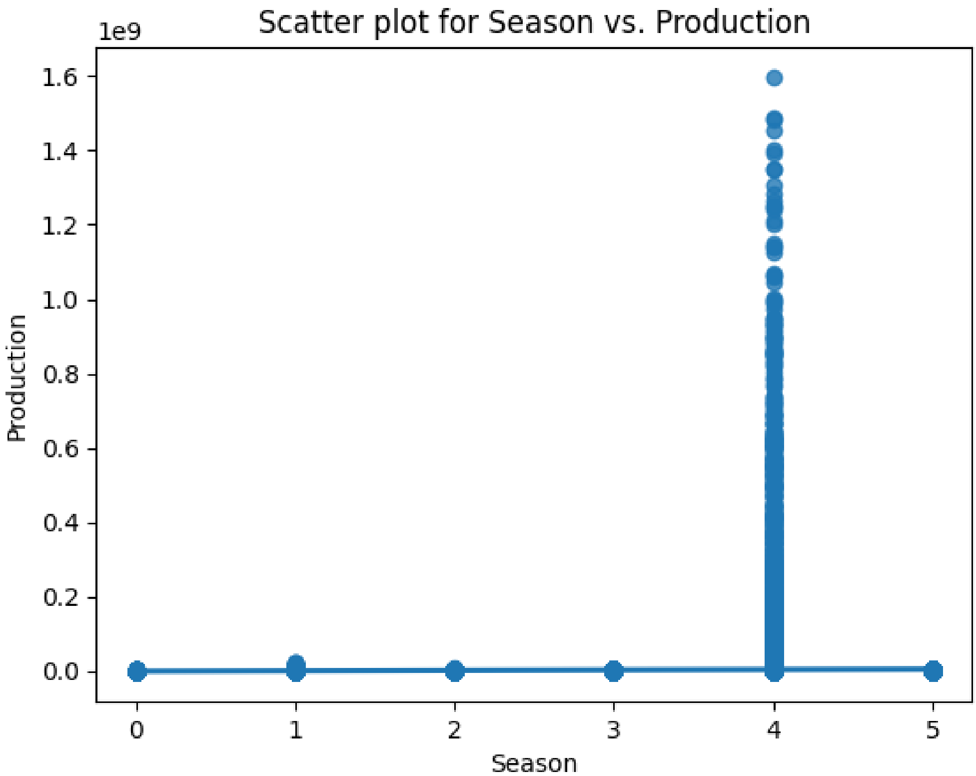
```
# it will generate scatter plots to visualize the relationship between each feature
x_axis=['State','District','Crop','Season','Yield','Area']
y_axis=df['Production']

for i in x_axis:
    sns.regplot(x=df[i], y=y_axis)
    plt.title(f'Scatter plot for {i} vs. Production')   #
    plt.xlabel(i)
    plt.ylabel('Production')
    plt.show()
```


Scatter plot for State vs. Production
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Production
0
5
10
15
20
25
30
35
State

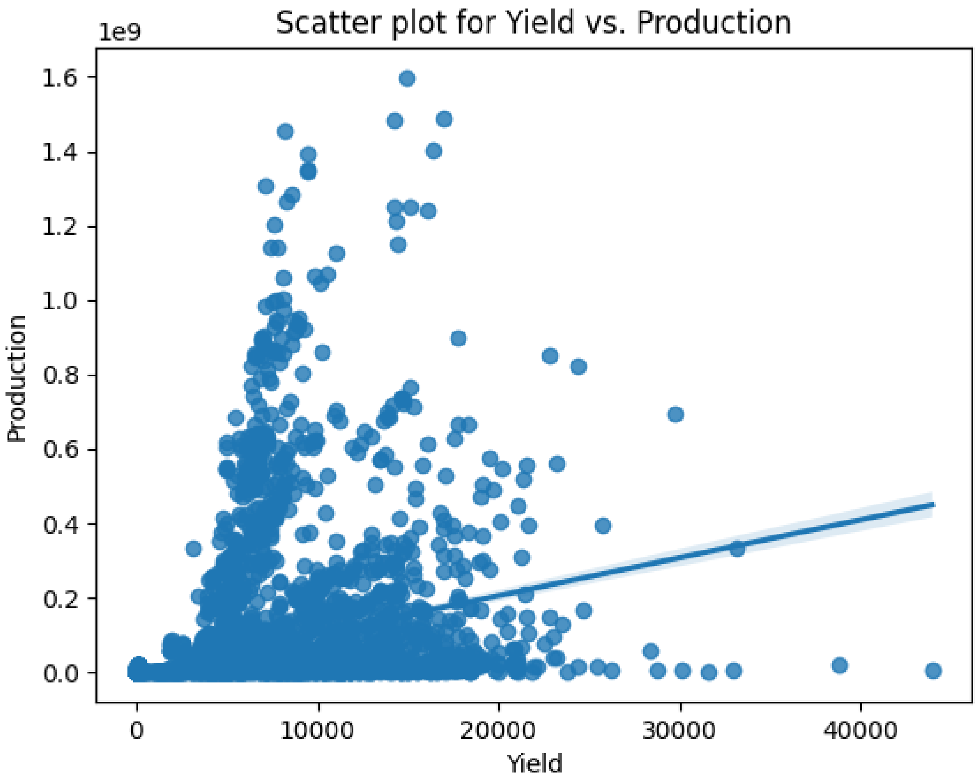

Scatter plot for District vs. Production
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Production
0
100
200
300
400
500
600
700
District

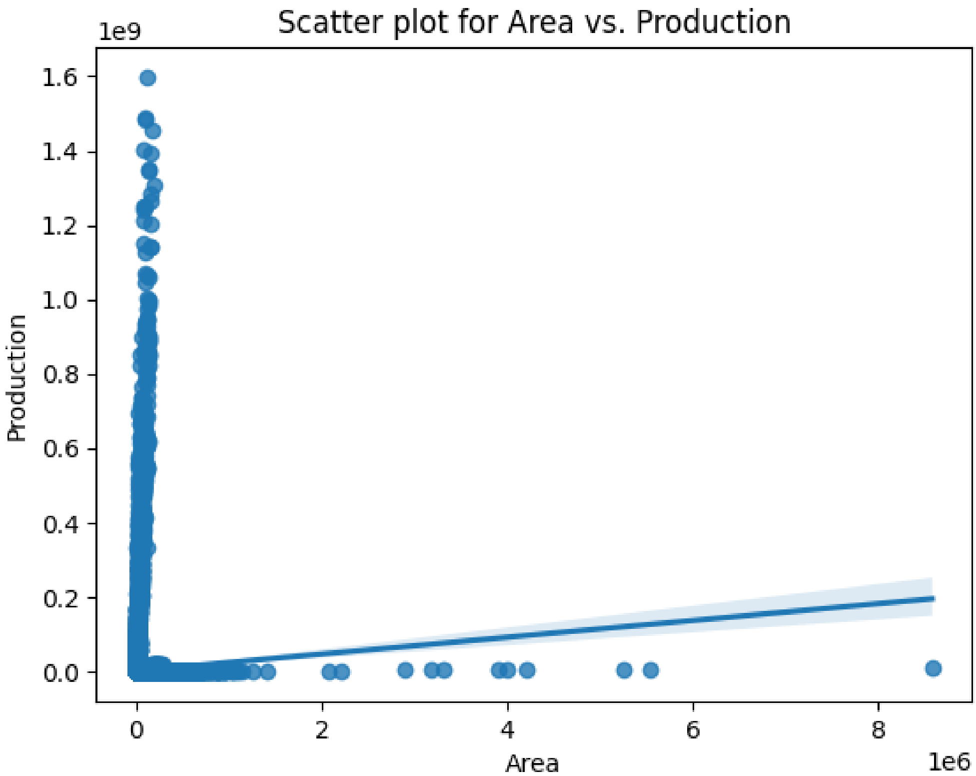


Scatter plot for Crop vs. Production
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Production
0
10
20
30
40
50
Crop



Scatter plot for Season vs. Production
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Production
0
1
2
3
4
5
Season

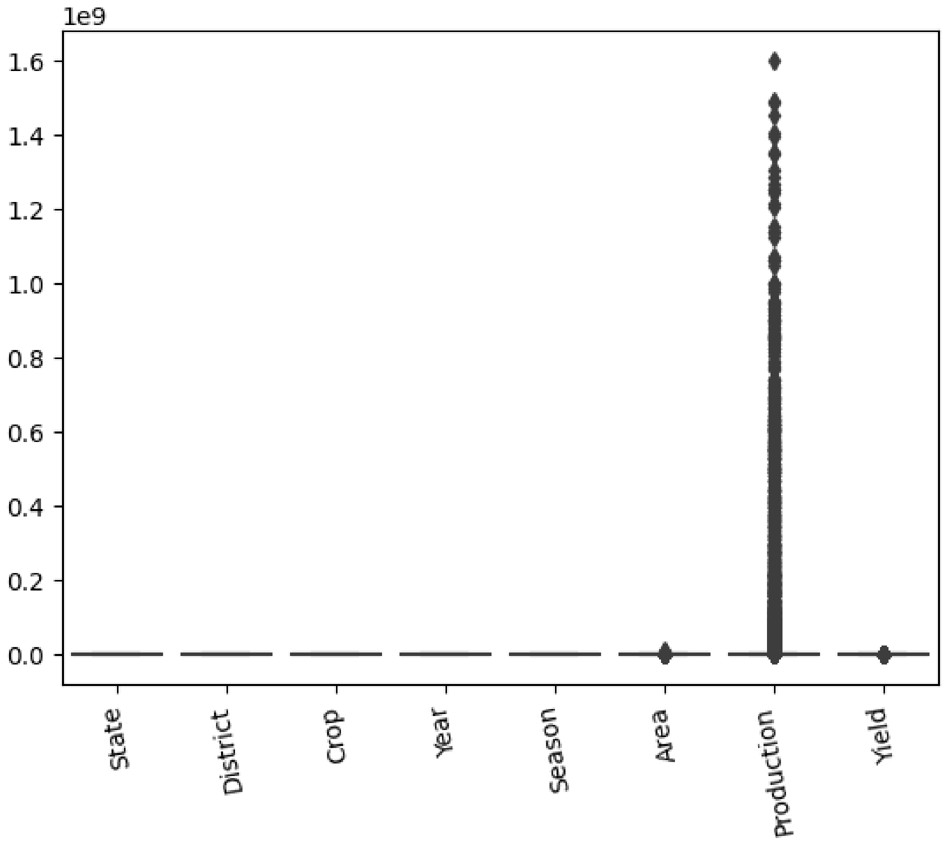




In [27]:
```
# to find outlayers
sns.boxplot(df)
plt.xticks(rotation=100)
```

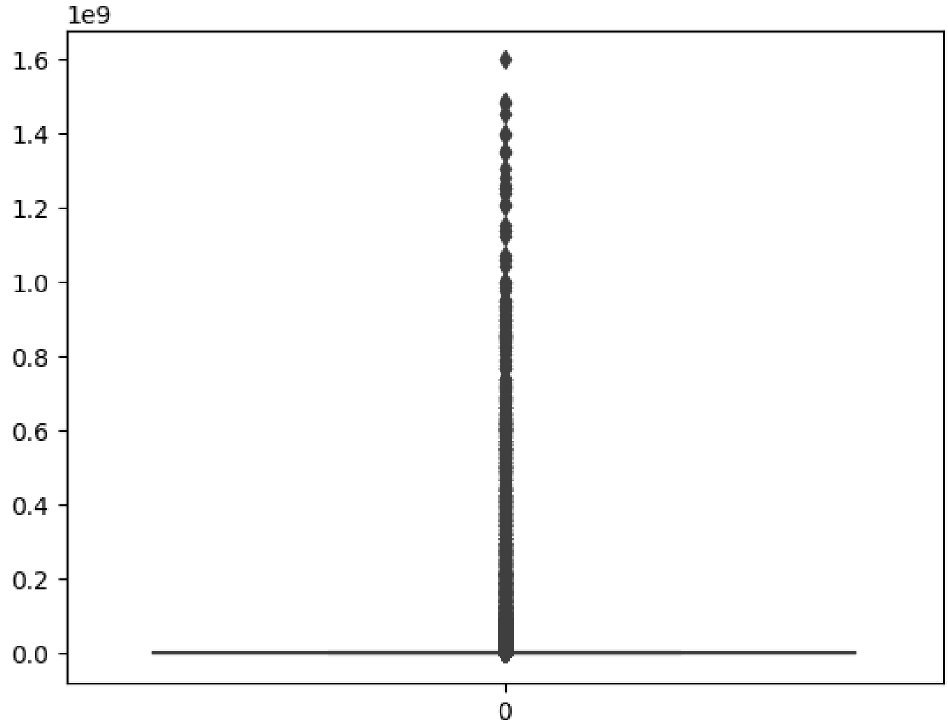
Out [27]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, 'State'),
  Text(1, 0, 'District'),
  Text(2, 0, 'Crop'),
```

```
Text(3, 0, 'Year'),
Text(4, 0, 'Season'),
Text(5, 0, 'Area'),
Text(6, 0, 'Production'),
Text(7, 0, 'Yield')])
```



In [28]:
```
sns.boxplot(df['Production'])
```

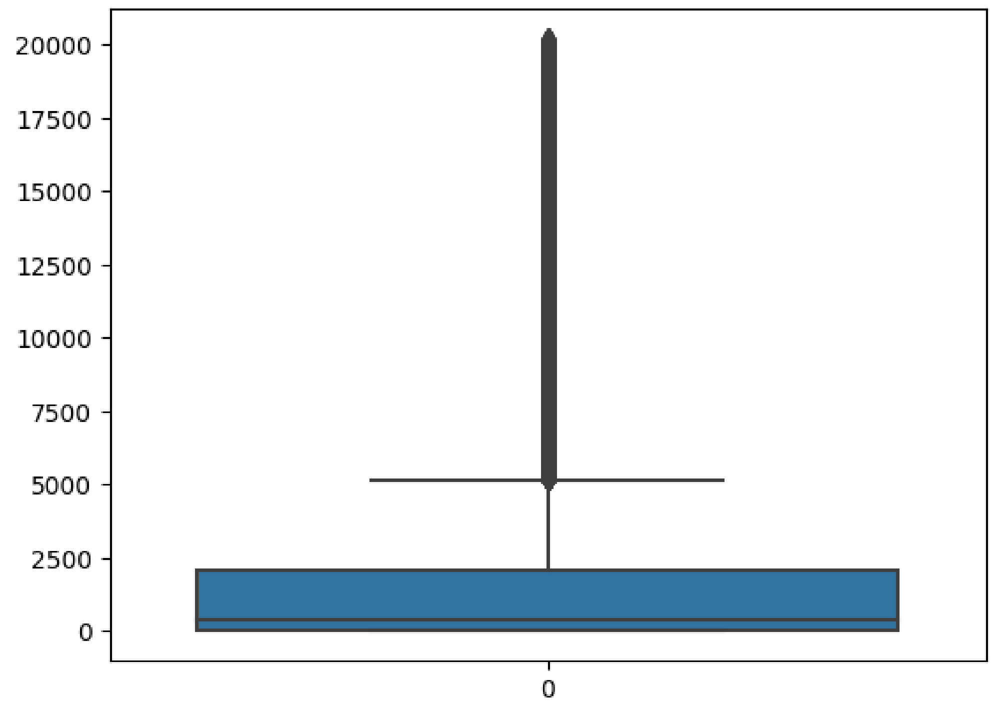
Out [28]: <Axes: >



In [29]:
```
# Calculate the IQR(inter quartile range)
Q1 = df['Production'].quantile(0.25)
Q3 = df['Production'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

outlier_mask = (df['Production'] < lower_bound) | (df['Production'] > upper_bound)

df2= df[~outlier_mask]
```

In [30]:
```
# after removing of outliers
sns.boxplot(df2['Production'])
```

Out [30]: <Axes: >



In [31]:
```
df2.shape
```

Out [31]: (283420, 8)

Feature selection using chi_square test

In [32]:
```
#feature slection is used to choose most relevent and informative features from the dataset

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
x=df2.drop(columns=['Production']).astype(int) #input feature
y=df2['Production'].astype(int)  #class label
k=5

selector=SelectKBest(chi2, k=k)

bst=selector.fit_transform(x, y)
bst

selected_feature_indices = selector.get_support(indices=True)

# Print the names of the selected features
selected_features = x.columns[selected_feature_indices]
print("Selected Features:", selected_features.tolist())
```

```
Selected Features: ['State', 'District', 'Crop', 'Area', 'Yield']
```

In [33]:
```
#convert training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(bst,y,test_size=0.30,random_state=42)
```

In [34]:
```
bst
```

Out [34]:
```
array([[   0,  481,    0, 1254,    1],
       [   0,  481,    0, 1258,    1],
       [   0,  481,    0, 1261,    1],
       ...,
       [  35,  531,   54, 3736,    1],
       [  35,  531,   54, 2752,    2],
       [  35,  531,   54, 2979,    2]])
```

In [35]:
```
y_test
```

Out [35]:
```
148391     4591
9951         36
174799     1951
40561       129
235883       36
          ...
190423       10
285242     1620
172472      403
76280     12377
208915     1106
Name: Production, Length: 85026, dtype: int64
```

Model creation

In [36]:
```
#multiple linear regression
from sklearn.linear_model import LinearRegression
model=LinearRegression()

# to get the default values for the parameters
model.get_params()
```

Out [36]:
```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
```

In [37]:
```
import warnings
warnings.filterwarnings('ignore')
```

Hyperparameter tuning

In [38]:
```
# hyperparmeter tuning to improve its performance

from sklearn.model_selection import GridSearchCV
# Define hyperparameter grid
parameter={'copy_X': [True,False], 'fit_intercept': [True,False], 'n_jobs': [None,1,5,7,6],

gsv=GridSearchCV(model,parameter,cv=10,scoring='accuracy')
gsv.fit(x_train,y_train)
# Access the best hyperparameters
gsv.best_params_
```

Out [38]:
```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': True}
```

Multiple linear regression model

In [39]:
```
# multiple linear regression model creation
model1=LinearRegression(positive=True)
```

```
model1.fit(x_train,y_train)
y_pred=model1.predict(x_test)
```

In [40]:

```
df3=pd.DataFrame({'actual_value':y_test,'predicted_value':y_pred,'difference':y_test-y_pred}
df3
```

Out [40]:

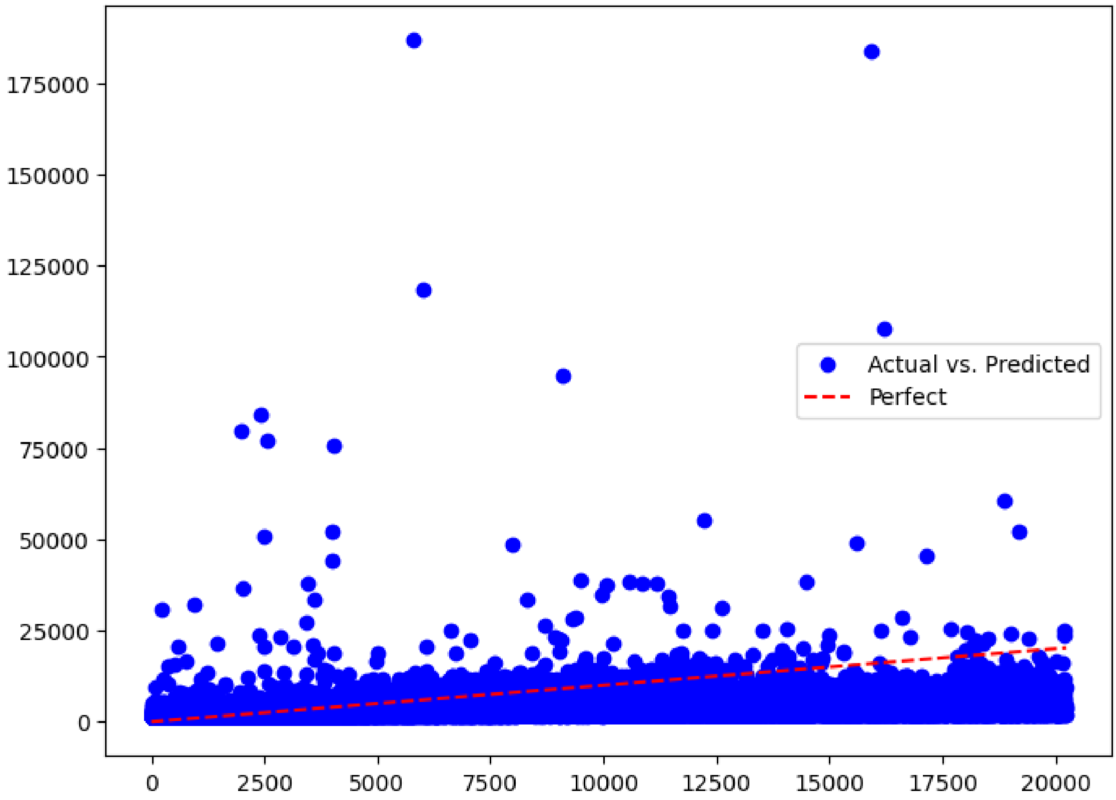
	actual_value	predicted_value	difference
148391	4591	1992.075132	2598.924868
9951	36	1594.119905	-1558.119905
174799	1951	2047.466262	-96.466262
40561	129	1565.272855	-1436.272855
235883	36	1549.396487	-1513.396487
...	...	...	...
190423	10	1637.541032	-1627.541032
285242	1620	1746.199486	-126.199486
172472	403	2235.325771	-1832.325771
76280	12377	3118.429979	9258.570021
208915	1106	2050.211662	-944.211662

85026 rows × 3 columns

In [41]:

```
plt.figure(figsize=(8, 6))
plt.scatter(df3['actual_value'], df3['predicted_value'], marker='o', color='blue', label='Ac
plt.plot([min(df3['actual_value']), max(df3['actual_value'])], [min(df3['actual_value']), ma
   linestyle='--', color='red', label='Perfect')
plt.legend()
```

Out [41]: <matplotlib.legend.Legend at 0x791b4f1882b0>



In [42]:
```
print('slpe is')
list(zip(x,model1.coef_))
```

```
slpe is
```

Out [42]:
```
[('State', 0.0),
 ('District', 0.20662190585144896),
 ('Crop', 0.0),
 ('Year', 0.23822819933439665),
 ('Season', 2.8536382665168345)]
```

In [43]:
```
print('constant is',model1.intercept_)
```

```
constant is 1506.6276641513823
```

In [44]:
```
#performance evaluation
#mean absolute error
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,mean_squared_(
r=r2_score(y_test,y_pred)
print('r2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
print('error percentage is',mean_absolute_percentage_error(y_test,y_pred))
```

```
r2 score 0.18563524207780147
MAE 2109.9434474906884
error percentage is 4.8343128553382984e+16
```

In [45]:
```
from sklearn.metrics import mean_squared_error
df1=mean_squared_error(y_test,y_pred)
df2=np.sqrt(df1)
df2
```

Out [45]: 3507.665909795286

Decision tree algorithm

In [50]:

```
# decision tree algorithm
from sklearn.tree import DecisionTreeRegressor
dec=DecisionTreeRegressor()
dec.fit(x_train,y_train)
y_pred1=dec.predict(x_test)
r1=r2_score(y_test,y_pred1)
print('r2 score',r2_score(y_test,y_pred1))
print('mean_absolute_percentage_error',mean_absolute_percentage_error(y_test,y_pred1))
```

```
r2 score 0.9015460124844398
mean_absolute_percentage_error 807076346319374.4
```

The decision tree regressor having mean_absolute_percentage_error is high so trying another regression model that is Random Forest regression model

> Indented block

Random forest algorithm

In [51]:

```
 #random forest regressor
from sklearn.ensemble import RandomForestRegressor
random=RandomForestRegressor()
random.fit(x_train,y_train)
y_pred2=random.predict(x_test)
r2=r2_score(y_test,y_pred2)
print('r2 score',r2_score(y_test,y_pred2))
print('mean_absolute_percentage_error ',mean_absolute_percentage_error(y_pred2,y_test))
```

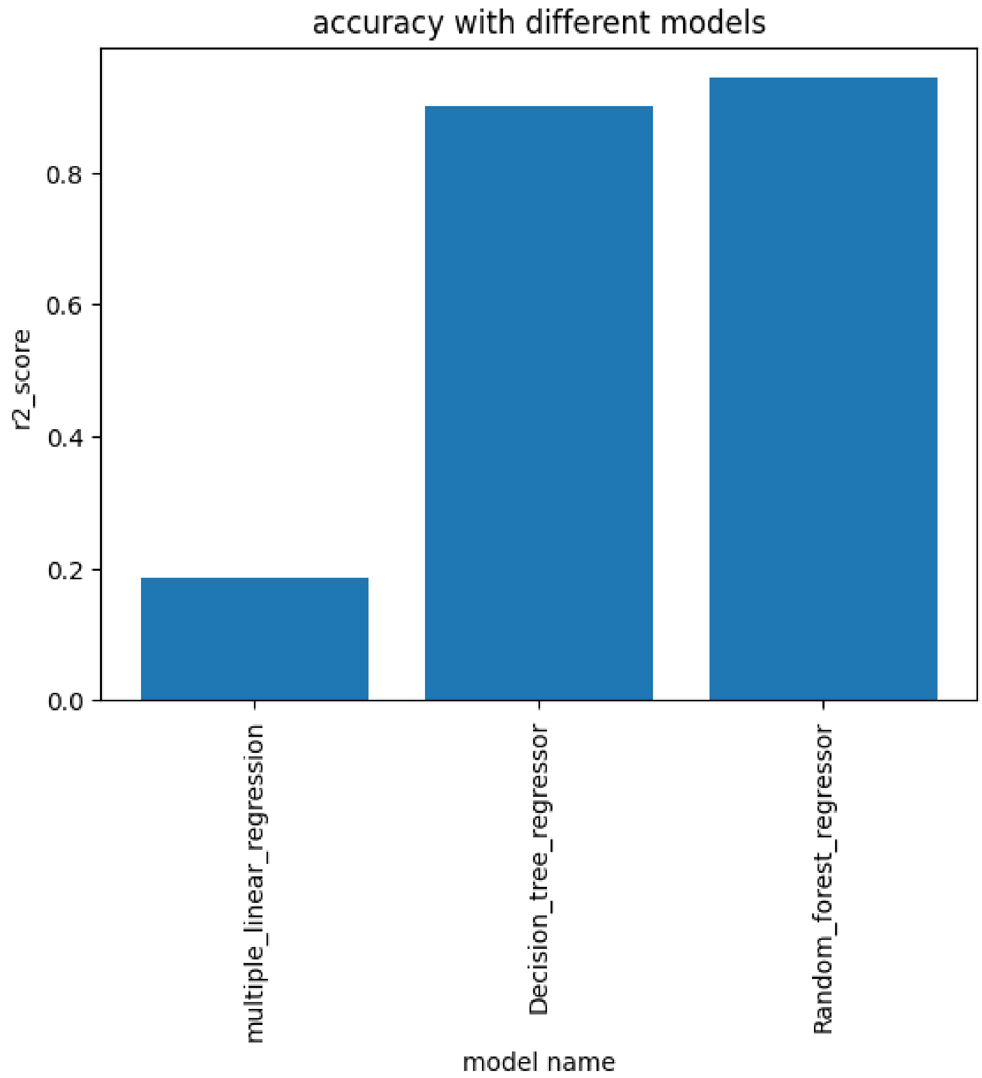
```
r2 score 0.9422896828462816
mean_absolute_percentage_error  0.19902025924091185
```

In [52]:

```
visual=['multiple_linear_regression','Decision_tree_regressor','Random_forest_regressor']
result=[r,r1,r2]
plt.bar(visual,result)
plt.xlabel('model name')
plt.ylabel('r2_score')
plt.title('accuracy with different models')
plt.xticks(rotation=90)
```

Out [52]:

```
([0, 1, 2],
 [Text(0, 0, 'multiple_linear_regression'),
  Text(1, 0, 'Decision_tree_regressor'),
  Text(2, 0, 'Random_forest_regressor')])
```



In [53]:
```
import pickle
with open('random_forest.pickle', 'wb') as dump_var:
     pickle.dump(random, dump_var)

pickle_in = open('random_forest.pickle', 'rb')
pickle_clf = pickle.load(pickle_in)

accuracy_pkl = pickle_clf.score(x_test,y_test)
accuracy_pkl
```

Out [53]: 0.9422896828462816

Conclusion

Among three models it shows that the Random forest regressor model is performing the best in terms of the R2 score and other .

==> The Random Forest model is the most accurate among these models, with a accuracy of 94%