Personal Challenge

Introduction:

This notebook contains modelling on the merged dataset of Crop production, Temperature in India and Rainfall in India All datasets were gotten from: https://data.gov.in/
Data transformation and exporting were done in two seperate notebooks
Data merging: Google cloud with Big query.
Data storing: Google cloud (cloud storage) and locally as csv files

Prediction goal: How much will be produced in tonnes i.e Production(tonne)

Prediction goal algorithm: Regression

In [1]:

```
import pandas as pd
import warnings

from sklearn.model_selection import train_test_split

from sklearn.feature_selection import f_classif
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest

from sklearn import linear_model
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

import numpy as np
from numpy import mean
from numpy import std
from numpy import absolute
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score


import seaborn as sns
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")
```

Importing rain dataset in india

In [2]:

```
raindata = pd.read_csv("rainfall_in_india_1901-2015.csv")
raindata
```

Out[2]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUA
0	ANDAMAN & NICOBAR ISLANDS	1901	49.2	87.1	29.2	2.3	528.8	517.5	365.1	481.1	332.6	388.5	558.2	33.6	3373

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUA
1	ANDAMAN & NICOBAR ISLANDS	1902	0.0	159.8	12.2	0.0	446.1	537.1	228.9	753.7	666.2	197.2	359.0	160.5	3520
2	ANDAMAN & NICOBAR ISLANDS	1903	12.7	144.0	0.0	1.0	235.1	479.9	728.4	326.7	339.0	181.2	284.4	225.0	2957
3	ANDAMAN & NICOBAR ISLANDS	1904	9.4	14.7	0.0	202.4	304.5	495.1	502.0	160.1	820.4	222.2	308.7	40.1	3079
4	ANDAMAN & NICOBAR ISLANDS	1905	1.3	0.0	3.3	26.9	279.5	628.7	368.7	330.5	297.0	260.7	25.4	344.7	2566
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4111	LAKSHADWEEP	2011	5.1	2.8	3.1	85.9	107.2	153.6	350.2	254.0	255.2	117.4	184.3	14.9	1533
4112	LAKSHADWEEP	2012	19.2	0.1	1.6	76.8	21.2	327.0	231.5	381.2	179.8	145.9	12.4	8.8	1405
4113	LAKSHADWEEP	2013	26.2	34.4	37.5	5.3	88.3	426.2	296.4	154.4	180.0	72.8	78.1	26.7	1426
4114	LAKSHADWEEP	2014	53.2	16.1	4.4	14.9	57.4	244.1	116.1	466.1	132.2	169.2	59.0	62.3	1395
4115	LAKSHADWEEP	2015	2.2	0.5	3.7	87.1	133.1	296.6	257.5	146.4	160.4	165.4	231.0	159.0	1642

4116 rows × 19 columns

Importing crop data containg crop information of

- When a crop was planted (per year and season)
- How much of a crop was produced in metrics tonnes
- How much land was used in the crop production in hectares
- The state which produced the crop
- The districts in the state
- The Average temperature it was during the period the crop was produced (grouped by the year and season)

In [3]:

```
cropproddata = pd.read_csv("crop_prod_final_dataset.csv")
cropproddata.drop("Unnamed: 0", axis = "columns", inplace = True)
cropproddata
```

Out[3]:

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)	Kh
0	Sikkim	EAST DISTRICT	2015	Wheat	143.0	162.0	Winter	21	
1	Sikkim	NORTH DISTRICT	2015	Wheat	50.0	45.0	Winter	21	
2	Sikkim	SOUTH DISTRICT	2015	Wheat	110.0	118.0	Winter	21	
3	Sikkim	WEST DISTRICT	2015	Wheat	20.0	21.0	Winter	21	
4	Odisha	BALESHWAR	2015	Jute	186.0	301.1	Autumn	27	
...	...	...	...	...	...	...	...	...	

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)	Kł
188964	Uttarakhand	PITHORAGARH	2000	Other Cereals & Millets	1827.0	457.0	Autumn	26	
188965	Uttarakhand	TEHRI GARHWAL	2000	Other Cereals & Millets	56.0	11.0	Autumn	26	
188966	Uttarakhand	UTTAR KASHI	2000	Other Cereals & Millets	626.0	219.0	Autumn	26	
188967	Puducherry	KARAIKAL	2000	Rice	16647.0	40159.0	Autumn	26	
188968	West Bengal	24 PARAGANAS SOUTH	2000	Rice	6927.0	12540.0	Autumn	26	

188969 rows × 15 columns

1. Data preparation on Rain data with Crop Production data

Getting the main columsn which I need from the rain data which are Subdivision, Year and Annual

- Annual: This is the total amount of rain which fell during the year
- Subdivision: The state which has the amount of rain

In [4]:

```
raindata = raindata[['SUBDIVISION','YEAR','ANNUAL']]
raindata
```

Out[4]:

	SUBDIVISION	YEAR	ANNUAL
0	ANDAMAN & NICOBAR ISLANDS	1901	3373.2
1	ANDAMAN & NICOBAR ISLANDS	1902	3520.7
2	ANDAMAN & NICOBAR ISLANDS	1903	2957.4
3	ANDAMAN & NICOBAR ISLANDS	1904	3079.6
4	ANDAMAN & NICOBAR ISLANDS	1905	2566.7
...	...	...	...
4111	LAKSHADWEEP	2011	1533.7
4112	LAKSHADWEEP	2012	1405.5
4113	LAKSHADWEEP	2013	1426.3
4114	LAKSHADWEEP	2014	1395.0
4115	LAKSHADWEEP	2015	1642.9

4116 rows × 3 columns

In the rainfall dataset, I noticed the subdivisions are in all caps, therefore I decided to convert the State names in Crop Production to uppercase

In [5]:
```
cropproddata["State_Name"] = cropproddata["State_Name"].str.upper()
cropproddata
```

Out[5]:

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
0	SIKKIM	EAST DISTRICT	2015	Wheat	143.0	162.0	Winter	21
1	SIKKIM	NORTH DISTRICT	2015	Wheat	50.0	45.0	Winter	21
2	SIKKIM	SOUTH DISTRICT	2015	Wheat	110.0	118.0	Winter	21
3	SIKKIM	WEST DISTRICT	2015	Wheat	20.0	21.0	Winter	21
4	ODISHA	BALESHWAR	2015	Jute	186.0	301.1	Autumn	27
...	...	...	...	...	...	...	...	...
188964	UTTARAKHAND	PITHORAGARH	2000	Other Cereals & Millets	1827.0	457.0	Autumn	26
188965	UTTARAKHAND	TEHRI GARHWAL	2000	Other Cereals & Millets	56.0	11.0	Autumn	26
188966	UTTARAKHAND	UTTAR KASHI	2000	Other Cereals & Millets	626.0	219.0	Autumn	26
188967	PUDUCHERRY	KARAIKAL	2000	Rice	16647.0	40159.0	Autumn	26
188968	WEST BENGAL	24 PARAGANAS SOUTH	2000	Rice	6927.0	12540.0	Autumn	26

188969 rows × 15 columns

In [6]:
```
raindata.rename(columns = {'SUBDIVISION' : "State_Name"}, inplace = True)
raindata.rename(columns = {'YEAR' : "Year"}, inplace = True)
raindata.rename(columns = {'ANNUAL' : "Rainfall"}, inplace = True)
raindata
```

Out[6]:

	State_Name	Year	Rainfall
0	ANDAMAN & NICOBAR ISLANDS	1901	3373.2
1	ANDAMAN & NICOBAR ISLANDS	1902	3520.7
2	ANDAMAN & NICOBAR ISLANDS	1903	2957.4
3	ANDAMAN & NICOBAR ISLANDS	1904	3079.6
4	ANDAMAN & NICOBAR ISLANDS	1905	2566.7
...	...	...	...
4111	LAKSHADWEEP	2011	1533.7
4112	LAKSHADWEEP	2012	1405.5
4113	LAKSHADWEEP	2013	1426.3

	State_Name	Year	Rainfall
4114	LAKSHADWEEP	2014	1395.0
4115	LAKSHADWEEP	2015	1642.9

4116 rows × 3 columns

Before merging, I decided to see if the amount of state and the state names both in rainfall and crop production dataset match

In [7]:

```
allrainstates = raindata["State_Name"].unique()
allrainstates
```

Out[7]:

```
array(['ANDAMAN & NICOBAR ISLANDS', 'ARUNACHAL PRADESH',
       'ASSAM & MEGHALAYA', 'NAGA MANI MIZO TRIPURA',
       'SUB HIMALAYAN WEST BENGAL & SIKKIM', 'GANGETIC WEST BENGAL',
       'ORISSA', 'JHARKHAND', 'BIHAR', 'EAST UTTAR PRADESH',
       'WEST UTTAR PRADESH', 'UTTARAKHAND', 'HARYANA DELHI & CHANDIGARH',
       'PUNJAB', 'HIMACHAL PRADESH', 'JAMMU & KASHMIR', 'WEST RAJASTHAN',
       'EAST RAJASTHAN', 'WEST MADHYA PRADESH', 'EAST MADHYA PRADESH',
       'GUJARAT REGION', 'SAURASHTRA & KUTCH', 'KONKAN & GOA',
       'MADHYA MAHARASHTRA', 'MATATHWADA', 'VIDARBHA', 'CHHATTISGARH',
       'COASTAL ANDHRA PRADESH', 'TELANGANA', 'RAYALSEEMA', 'TAMIL NADU',
       'COASTAL KARNATAKA', 'NORTH INTERIOR KARNATAKA',
       'SOUTH INTERIOR KARNATAKA', 'KERALA', 'LAKSHADWEEP'], dtype=object)
```

In [8]:

```
cropproddata["State_Name"].unique()
```

Out[8]:

```
array(['SIKKIM', 'ODISHA', 'MANIPUR', 'KARNATAKA', 'PUDUCHERRY', 'ASSAM',
       'BIHAR', 'GUJARAT', 'KERALA', 'MAHARASHTRA', 'MEGHALAYA',
       'UTTAR PRADESH', 'UTTARAKHAND', 'WEST BENGAL', 'NAGALAND',
       'TRIPURA', 'HARYANA', 'HIMACHAL PRADESH', 'PUNJAB', 'RAJASTHAN',
       'ANDHRA PRADESH', 'CHHATTISGARH', 'DADRA AND NAGAR HAVELI', 'GOA',
       'JHARKHAND', 'MADHYA PRADESH', 'TAMIL NADU', 'TELANGANA ',
       'ANDAMAN AND NICOBAR ISLANDS', 'ARUNACHAL PRADESH', 'CHANDIGARH',
       'JAMMU AND KASHMIR ', 'MIZORAM'], dtype=object)
```

After carefully looking through each state, I noticed some states in the rain fall data are not in crop produciton and some states in rainfall data are combined to one state and not split like in Crop production dataset. Therefore I have to transform the dataset in rainfall to match crop production before merging

1a. Data transformation

In [9]:

```
cropproddata = cropproddata.replace("JAMMU AND KASHMIR ", "JAMMU AND KASHMIR")
```

List of all states in rainfall and what needs to be changed or done for it to match crop prod states

- ANDAMAN & NICOBAR ISLANDS,
- ARUNACHAL PRADESH,
- ASSAM & MEGHALAYA (cropproddata: ASSAM, MEGHLAYA)
- NAGA MANI MIZO TRIPURA (cropproddata: NAGALAND, TRIPURA)
- SUB HIMALAYAN WEST BENGAL & SIKKIM (cropproddata: SIKKIM)
- GANGETIC WEST BENGAL(cropproddata: WEST BENGAL)
- ORISSA (cropproddata: ODISHA)

- JHARKHAND
- BIHAR
- EAST UTTAR PRADESH (cropproddata: UTTAR PRADESH)
- WEST UTTAR PRADESH (cropproddata: UTTAR PRADESH)
- UTTARAKHAND
- HARYANA DELHI & CHANDIGARH: (croppoddata: HARYANA, CHANDIGARH)
- PUNJAB
- HIMACHAL PRADESH
- JAMMU & KASHMIR (cropproddata: JAMMU AND KASHMIR)
- WEST RAJASTHAN (cropproddata: RAJASTHAN)
- EAST RAJASTHAN (cropproddata: RAJASTHAN)
- WEST MADHYA PRADESH (cropproddata: MADHYA PRADESH)#####
- EAST MADHYA PRADESH (cropproddata: MADHYA PRADESH)
- GUJARAT REGION (cropproddata: GUJARAT)
- KONKAN & GOA (cropproddata: GOA)
- MADHYA MAHARASHTRA (cropproddata: MAHARASHTRA)
- MATATHWADA (nothing)
- VIDARBHA (nothing)
- CHHATTISGARH
- COASTAL ANDHRA PRADESH (cropproddata: ANDHRA PRADESH)
- TELANGANA
- RAYALSEEMA (nothing)
- TAMIL NADU
- COASTAL KARNATAKA (cropproddata: KARNATAKA)
- NORTH INTERIOR KARNATAKA (cropproddata: KARNATAKA)
- SOUTH INTERIOR KARNATAKA (cropproddata: KARNATAKA)
- KERALA
- LAKSHADWEEP (nothing)

Dropping all irrelevant state values in rainfall data and Replacing state names in raindata to make merging easier

In [10]:

```
def dropStates(datadf,statename):
    indexNames = datadf[datadf['State_Name'] == statename].index
    datadf.drop(indexNames, inplace = True)
```

In [11]:

```
def replaceStateName(datadf, prevstatename, newstatename):
    datadf["State_Name"].replace(prevstatename, newstatename, inplace = True)
```

In [12]:

```
dropStates(raindata,"MATATHWADA")
dropStates(raindata,"VIDARBHA")
```

```
dropStates(raindata,"RAYALSEEMA")
dropStates(raindata,"LAKSHADWEEP")

replaceStateName(raindata, "ANDAMAN & NICOBAR ISLANDS", "ANDAMAN AND NICOBAR ISLANDS")
replaceStateName(raindata, "JAMMU & KASHMIR", "JAMMU AND KASHMIR")
replaceStateName(raindata, "SUB HIMALAYAN WEST BENGAL & SIKKIM", "SIKKIM")
replaceStateName(raindata, "GANGETIC WEST BENGAL", "WEST BENGAL")
replaceStateName(raindata, "ORISSA", "ODISHA")
replaceStateName(raindata, "GUJARAT REGION", "GUJARAT")
replaceStateName(raindata, "KONKAN & GOA", "GOA")
replaceStateName(raindata, "MADHYA MAHARASHTRA", "MAHARASHTRA")
replaceStateName(raindata, "COASTAL ANDHRA PRADESH", "ANDHRA PRADESH")
```

Rainfall Dataframe after replacing and removing irrelevant states

In [13]:

```
raindata
```

Out[13]:

	State_Name	Year	Rainfall
0	ANDAMAN AND NICOBAR ISLANDS	1901	3373.2
1	ANDAMAN AND NICOBAR ISLANDS	1902	3520.7
2	ANDAMAN AND NICOBAR ISLANDS	1903	2957.4
3	ANDAMAN AND NICOBAR ISLANDS	1904	3079.6
4	ANDAMAN AND NICOBAR ISLANDS	1905	2566.7
...	...	...	...
3997	KERALA	2011	3035.1
3998	KERALA	2012	2151.1
3999	KERALA	2013	3255.4
4000	KERALA	2014	3046.4
4001	KERALA	2015	2600.6

3657 rows × 3 columns

Getting a dataframe for each of the states whose names are considered as one in crop production dataset to work on them individually by findiing the average rainfall for that region and given the average rainfall to that region regardless of which part of the state the region recides

In [14]:

```
filter_list = ['EAST UTTAR PRADESH', 'WEST UTTAR PRADESH']
east_para = raindata[raindata.State_Name.isin(filter_list)]

test = east_para.groupby(["Year"])["Rainfall"].mean()
test = test.to_frame()
test = test.reset_index()
```

In [15]:

```
filter_list = ['EAST RAJASTHAN', 'WEST RAJASTHAN']
er_para = raindata[raindata.State_Name.isin(filter_list)]

er = er_para.groupby(["Year"])["Rainfall"].mean()
er = er.to_frame()
er = er.reset_index()
```

In [16]:

```
filter_list = ['EAST MADHYA PRADESH', 'WEST MADHYA PRADESH']
mp_para = raindata[raindata.State_Name.isin(filter_list)]

mp = mp_para.groupby(["Year"])["Rainfall"].mean()
mp = mp.to_frame()
mp = mp.reset_index()
```

In [17]:

```
filter_list = ['COASTAL KARNATAKA', 'NORTH INTERIOR KARNATAKA', "SOUTH INTERIOR KARNATAKA'
k_para = raindata[raindata.State_Name.isin(filter_list)]

k = k_para.groupby(["Year"])["Rainfall"].mean()
k = k.to_frame()
k = k.reset_index()
```

In [18]:

```
merged_inner = pd.merge(left=east_para, right=test, left_on='Year', right_on='Year')
merged_inner.drop("Rainfall_x", axis = "columns", inplace = True)

merged_inner_er = pd.merge(left=er_para, right=er, left_on='Year', right_on='Year')
merged_inner_er.drop("Rainfall_x", axis = "columns", inplace = True)

merged_inner_mp = pd.merge(left=mp_para, right=mp, left_on='Year', right_on='Year')
merged_inner_mp.drop("Rainfall_x", axis = "columns", inplace = True)

merged_inner_k = pd.merge(left=k_para, right=k, left_on='Year', right_on='Year')
merged_inner_k.drop("Rainfall_x", axis = "columns", inplace = True)
```

In [19]:

```
merged_inner["State_Name"].replace("EAST UTTAR PRADESH", "UTTAR PRADESH", inplace = True)
merged_inner["State_Name"].replace("WEST UTTAR PRADESH", "UTTAR PRADESH", inplace = True)

merged_inner_er["State_Name"].replace("EAST RAJASTHAN", "RAJASTHAN", inplace = True)
merged_inner_er["State_Name"].replace("WEST RAJASTHAN", "RAJASTHAN", inplace = True)

merged_inner_mp["State_Name"].replace("EAST MADHYA PRADESH", "MADHYA PRADESH", inplace = T
merged_inner_mp["State_Name"].replace("WEST MADHYA PRADESH", "MADHYA PRADESH", inplace = T

merged_inner_k["State_Name"].replace("COASTAL KARNATAKA", "KARNATAKA", inplace = True)
merged_inner_k["State_Name"].replace("NORTH INTERIOR KARNATAKA", "KARNATAKA", inplace = Tr
merged_inner_k["State_Name"].replace("SOUTH INTERIOR KARNATAKA", "KARNATAKA", inplace = Tr
```

In [20]:

```
merged_inner.rename(columns = {'Rainfall_y' : 'Rainfall'}, inplace = True)
merged_inner_er.rename(columns = {'Rainfall_y' : 'Rainfall'}, inplace = True)
merged_inner_mp.rename(columns = {'Rainfall_y' : 'Rainfall'}, inplace = True)
merged_inner_k.rename(columns = {'Rainfall_y' : 'Rainfall'}, inplace = True)
```

Removing duplicated values from merged tables

In [21]:

```
merged_inner.drop_duplicates(inplace = True)
merged_inner_er.drop_duplicates(inplace = True)
merged_inner_mp.drop_duplicates(inplace = True)
merged_inner_k.drop_duplicates(inplace = True)
```

Dropping old values for the regions I worked on above

In [22]:

```
indexNames = raindata[raindata['State_Name'] == 'EAST UTTAR PRADESH'].index
indexNames2 = raindata[raindata['State_Name'] == 'WEST UTTAR PRADESH'].index
indexNames3 = raindata[raindata['State_Name'] == 'EAST RAJASTHAN'].index
indexNames4 = raindata[raindata['State_Name'] == 'WEST RAJASTHAN'].index
indexNames5 = raindata[raindata['State_Name'] == 'EAST MADHYA PRADESH'].index
```

```
indexNames6 = raindata[raindata['State_Name'] == 'WEST MADHYA PRADESH'].index
indexNames7 = raindata[raindata['State_Name'] == 'COASTAL KARNATAKA'].index
indexNames8 = raindata[raindata['State_Name'] == 'NORTH INTERIOR KARNATAKA'].index
indexNames9 = raindata[raindata['State_Name'] == 'SOUTH INTERIOR KARNATAKA'].index

raindata= raindata.drop(indexNames)
raindata = raindata.drop(indexNames2)
raindata= raindata.drop(indexNames3)
raindata = raindata.drop(indexNames4)
raindata= raindata.drop(indexNames5)
raindata = raindata.drop(indexNames6)
raindata= raindata.drop(indexNames7)
raindata = raindata.drop(indexNames8)
raindata= raindata.drop(indexNames9)
```

Rainfall dataframe without the old regions(West, East etc.)

In [23]:

```
raindata
```

Out[23]:

	State_Name	Year	Rainfall
0	ANDAMAN AND NICOBAR ISLANDS	1901	3373.2
1	ANDAMAN AND NICOBAR ISLANDS	1902	3520.7
2	ANDAMAN AND NICOBAR ISLANDS	1903	2957.4
3	ANDAMAN AND NICOBAR ISLANDS	1904	3079.6
4	ANDAMAN AND NICOBAR ISLANDS	1905	2566.7
...	...	...	...
3997	KERALA	2011	3035.1
3998	KERALA	2012	2151.1
3999	KERALA	2013	3255.4
4000	KERALA	2014	3046.4
4001	KERALA	2015	2600.6

2622 rows × 3 columns

Merging average values of the deleted regions to rainfall data

In [24]:

```
raindata = raindata.append(merged_inner)
raindata = raindata.append(merged_inner_er)
raindata = raindata.append(merged_inner_mp)
raindata = raindata.append(merged_inner_k)
```

New rainfall dataframe

In [25]:

```
raindata.rename(columns = {'Year' : "Crop_Year"}, inplace = True)
raindata
```

Out[25]:

	State_Name	Crop_Year	Rainfall
0	ANDAMAN AND NICOBAR ISLANDS	1901	3373.200000
1	ANDAMAN AND NICOBAR ISLANDS	1902	3520.700000

	State_Name	Crop_Year	Rainfall
2	ANDAMAN AND NICOBAR ISLANDS	1903	2957.400000
3	ANDAMAN AND NICOBAR ISLANDS	1904	3079.600000
4	ANDAMAN AND NICOBAR ISLANDS	1905	2566.700000
...	...	...	...
330	KARNATAKA	2011	1883.733333
333	KARNATAKA	2012	730.800000
336	KARNATAKA	2013	2021.100000
339	KARNATAKA	2014	1875.133333
342	KARNATAKA	2015	1590.133333

3082 rows × 3 columns

Merging Crop production and Rainfall dataframe

In [26]:

```
df_merge = pd.merge(cropproddata, raindata, on=['State_Name', 'Crop_Year'], how='left')
```

In [27]:

```
df_merge
```

Out[27]:

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
0	SIKKIM	EAST DISTRICT	2015	Wheat	143.0	162.0	Winter	21
1	SIKKIM	NORTH DISTRICT	2015	Wheat	50.0	45.0	Winter	21
2	SIKKIM	SOUTH DISTRICT	2015	Wheat	110.0	118.0	Winter	21
3	SIKKIM	WEST DISTRICT	2015	Wheat	20.0	21.0	Winter	21
4	ODISHA	BALESHWAR	2015	Jute	186.0	301.1	Autumn	27
...	...	...	...	...	...	...	...	...
188964	UTTARAKHAND	PITHORAGARH	2000	Other Cereals & Millets	1827.0	457.0	Autumn	26
188965	UTTARAKHAND	TEHRI GARHWAL	2000	Other Cereals & Millets	56.0	11.0	Autumn	26
188966	UTTARAKHAND	UTTAR KASHI	2000	Other Cereals & Millets	626.0	219.0	Autumn	26
188967	PUDUCHERRY	KARAIKAL	2000	Rice	16647.0	40159.0	Autumn	26
188968	WEST BENGAL	24 PARAGANAS SOUTH	2000	Rice	6927.0	12540.0	Autumn	26

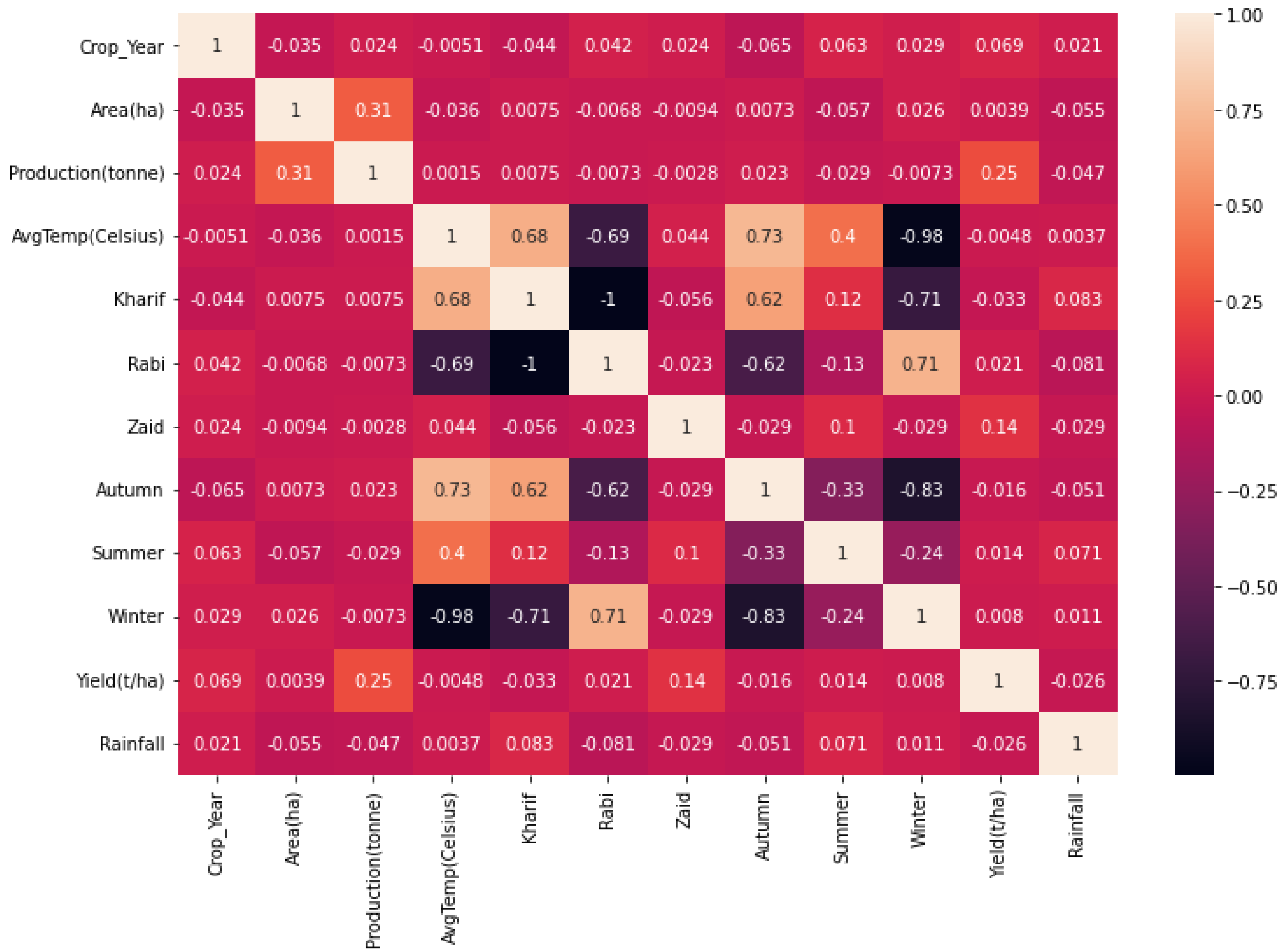
188969 rows × 16 columns

In [28]:
```
df_merge= df_merge.dropna()
```

Checking correlation matrix

In [29]:
```
plt.subplots(figsize=(12,8))
sns.heatmap(df_merge.corr(), annot = True)
```

Out[29]: <AxesSubplot:>



Hypothesis:

With the addition of rain dataframe, I thought the amount of rainfall should have a good correlation with how much was produced in tonnes.

Conclusion:

From the correlation matrix, I can see that only Area has a good correlation with Production(tonne) which is really bad, therefore I have to see why rainfall doesnt have a good correlation with production(tonne)

Note:

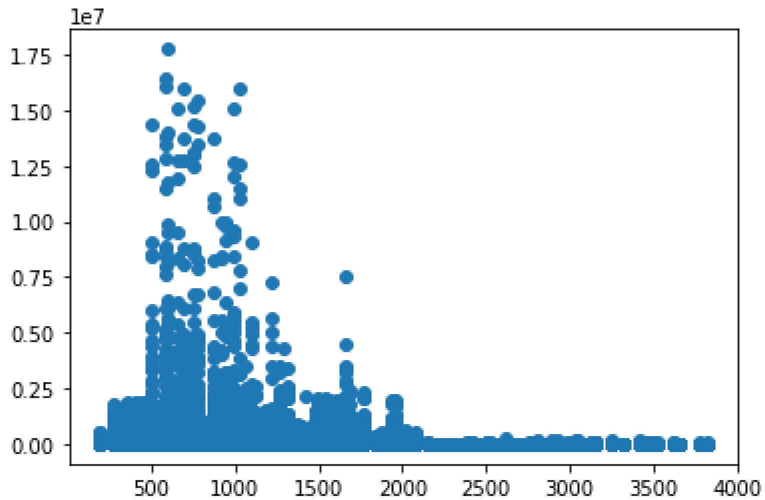
I will not be taken into account Yield(t/ha), this is because this column was generated for the modelling regarding time series and not for this notebook

Visualization of Rainfall and Crop Production

Plotting a scatter plot to see if there is any correlation between Rainfall and Production(tonne)

In [30]:
```
plt.scatter(df_merge['Rainfall'], df_merge['Production(tonne)'])
```

Out[30]: <matplotlib.collections.PathCollection at 0x21b17d31940>



Form the above graph, I can see that there are alot of scatter plots and some might be outliers, therefore I beleive maybe this is why rainfall, avgtemp and other features dont have a good correlation with production

I decided to check and remove any of the values of Production(tonne) equal 0.0, because this for my modelling is considered an outlier

In [31]:
```
indexNames = df_merge[df_merge['Production(tonne)'] == 0.0].index
df_merge.drop(indexNames, inplace = True)
```

In [32]:
```
df_merge[df_merge['Production(tonne)'] == 0.0]
```

Out[32]:

State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)	Kharif	Rab

In [33]:
```
df_merge
```

Out[33]:

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
0	SIKKIM	EAST DISTRICT	2015	Wheat	143.0	162.0	Winter	21
1	SIKKIM	NORTH DISTRICT	2015	Wheat	50.0	45.0	Winter	21
2	SIKKIM	SOUTH DISTRICT	2015	Wheat	110.0	118.0	Winter	21
3	SIKKIM	WEST DISTRICT	2015	Wheat	20.0	21.0	Winter	21
4	ODISHA	BALESHWAR	2015	Jute	186.0	301.1	Autumn	27
...	...	...	...	...	...	...	...	...

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
188963	UTTARAKHAND	PAURI GARHWAL	2000	Other Cereals & Millets	634.0	159.0	Autumn	26
188964	UTTARAKHAND	PITHORAGARH	2000	Other Cereals & Millets	1827.0	457.0	Autumn	26
188965	UTTARAKHAND	TEHRI GARHWAL	2000	Other Cereals & Millets	56.0	11.0	Autumn	26
188966	UTTARAKHAND	UTTAR KASHI	2000	Other Cereals & Millets	626.0	219.0	Autumn	26
188968	WEST BENGAL	24 PARAGANAS SOUTH	2000	Rice	6927.0	12540.0	Autumn	26

158852 rows × 16 columns

Here I decided to drop the Yield column because I do not need this column for my predictive problem

In [34]:

```
df_merge2 = df_merge.drop(["Yield(t/ha)"], axis = "columns")
df_merge2
```

Out[34]:

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
0	SIKKIM	EAST DISTRICT	2015	Wheat	143.0	162.0	Winter	21
1	SIKKIM	NORTH DISTRICT	2015	Wheat	50.0	45.0	Winter	21
2	SIKKIM	SOUTH DISTRICT	2015	Wheat	110.0	118.0	Winter	21
3	SIKKIM	WEST DISTRICT	2015	Wheat	20.0	21.0	Winter	21
4	ODISHA	BALESHWAR	2015	Jute	186.0	301.1	Autumn	27
...	...	...	...	...	...	...	...	...
188963	UTTARAKHAND	PAURI GARHWAL	2000	Other Cereals & Millets	634.0	159.0	Autumn	26
188964	UTTARAKHAND	PITHORAGARH	2000	Other Cereals & Millets	1827.0	457.0	Autumn	26
188965	UTTARAKHAND	TEHRI GARHWAL	2000	Other Cereals & Millets	56.0	11.0	Autumn	26

	State_Name	District_Name	Crop_Year	Crop	Area(ha)	Production(tonne)	Season	AvgTemp(Celsius)
188966	UTTARAKHAND	UTTAR KASHI	2000	Other Cereals & Millets	626.0	219.0	Autumn	26
188968	WEST BENGAL	24 PARAGANAS SOUTH	2000	Rice	6927.0	12540.0	Autumn	26

158852 rows × 15 columns

1b. Outlier Detection

Getting the total amount produced for each state over the years

In [35]:

```
df_merged_states = df_merge2.groupby(["State_Name", "Crop_Year", "Crop",  "Season", "AvgTe
df_merged_states = df_merged_states.reset_index()
df_merged_states
```

Out[35]:

	State_Name	Crop_Year	Crop	Season	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winter
0	ANDAMAN AND NICOBAR ISLANDS	2000	Arecanut	Autumn	26	0	1	0	1	0	0
1	ANDAMAN AND NICOBAR ISLANDS	2000	Other Kharif pulses	Autumn	26	1	0	0	1	0	0
2	ANDAMAN AND NICOBAR ISLANDS	2000	Rice	Autumn	26	1	0	0	1	0	0
3	ANDAMAN AND NICOBAR ISLANDS	2001	Arecanut	Autumn	26	0	1	0	1	0	0
4	ANDAMAN AND NICOBAR ISLANDS	2001	Other Kharif pulses	Autumn	26	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
8318	WEST BENGAL	2014	Soyabean	Autumn	26	1	0	0	1	0	0
8319	WEST BENGAL	2014	Sunflower	Autumn	26	1	0	0	1	0	0
8320	WEST BENGAL	2014	Urad	Autumn	26	1	0	0	1	0	0
8321	WEST BENGAL	2014	Urad	Winter	20	1	0	0	0	0	1
8322	WEST BENGAL	2014	Wheat	Winter	20	0	1	0	0	0	1

8323 rows × 14 columns

In [36]:

```
df_merged_states.describe()
```

Out[36]:

	Crop_Year	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winter
count	8323.000000	8323.000000	8323.000000	8323.000000	8323.000000	8323.000000	8323.000000	8323.000000
mean	2005.866514	24.125315	0.727382	0.271777	0.000841	0.542232	0.079298	0.378469
std	5.023433	2.722182	0.445333	0.444902	0.028990	0.498243	0.270220	0.485035
min	1997.000000	20.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2002.000000	21.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	2006.000000	26.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000
75%	2010.000000	26.000000	1.000000	1.000000	0.000000	1.000000	0.000000	1.000000
max	2015.000000	28.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

From the describe I noticed the max value of crop Production(tonne) is 146,576,800tonnes, therefore I decided to look for what crop this exactly is

In [37]:

```
df_merged_states[df_merged_states["Production(tonne)"] > 140000000]
```

Out[37]:

	State_Name	Crop_Year	Crop	Season	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winter
7080	UTTAR PRADESH	2013	Sugarcane	Autumn	26	1	0	0	1	0	0
7122	UTTAR PRADESH	2014	Sugarcane	Autumn	26	1	0	0	1	0	0

From the above, the crop which has its Production(tonne) around 140,000,000tonnes is Sugarcane.

Hypothesis:

I beleive this value is really large and I decided to research if Uttar pradesh a state in India really produced this amount of sugar cane in the year 2013 and 2014.

Conclusion:

From my research, I can say that the values of crop production of sugar cane is inaccurate, this is because according to The Economic Times news website, the highest amount of sugar cane India produced during the period 2013 and 2014 is 24.3million tonnes and not above this mark (Website link: https://economictimes.indiatimes.com/news/economy/agriculture/sugar-output-to-rise-by-4-to-25-3-million-tonnes-in-2014-15/articleshow/38210396.cms?from=mdr#:~:text=Sugar%20production%20is%20estimated%20at%2024.3%20million%20tonnes%20in%20the%20
).

Research question/thoughts:

With the result from my research I decided to look into what was the highest ever recorded amount produced of a crop in india, thus I would know what the maximum value for production(tonnes) should be, thus helping me with detecting outliers.

Conclusion:

After a good amount of research, I found from the website "Zeenews" which states the Highest and Lowest production of food grain in India for 5 states. (website link: https://zeenews.india.com/economy/states-with-highest-and-lowest-production-of-foodgrains-in-last-4-years-2190600.html#:~:text=States%20with%20Highest%20Production%20Of%20Foodgrains%20%2D%20Total%20Foodg)

I decided to take the highest amount ever recorded between 2013-2015(50, 027, 000) and the lowest recorded 2013-2015(50,000 tonnes) as my minimum and max thresholds to remove my outliers

In [38]:
```
max_threshold = 50000000.0
min_threshold = 50000.0
```

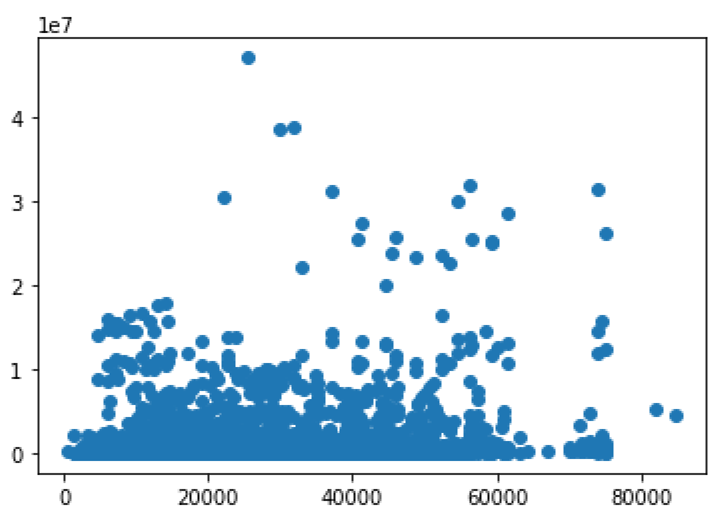
In [39]:
```
df_merged_states_refined = df_merged_states[(df_merged_states["Production(tonne)"]<max_thr
df_merged_states_refined = df_merged_states_refined.reset_index()
df_merged_states_refined = df_merged_states_refined.drop("index", axis = "columns")
```

Visualization of Rainfall and Crop Production

Plotting another scatter plot to see if there is any correlation between Rainfall and Production(tonne) after removal of outliers

In [40]:
```
plt.scatter(df_merged_states_refined['Rainfall'], df_merged_states_refined['Production(ton
```

Out[40]:
```
<matplotlib.collections.PathCollection at 0x21b17bc4cd0>
```



After the removal of the outliers, I can say from the above graph rainfall has a better correlation with crop production and the values are better seen

Reordering Column names

In [41]:
```
df_merged_states_refined = df_merged_states_refined[["State_Name", "Crop", "Season", "Crop
df_merged_states_refined
```

Out[41]:

	State_Name	Crop	Season	Crop_Year	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winte
0	ANDHRA PRADESH	Bajra	Autumn	1997	26	1	0	0	1	0	

	State_Name	Crop	Season	Crop_Year	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winte
1	ANDHRA PRADESH	Cotton(lint)	Autumn	1997	26	1	0	0	1	0	
2	ANDHRA PRADESH	Dry chillies	Autumn	1997	26	1	0	0	1	0	
3	ANDHRA PRADESH	Gram	Winter	1997	20	0	1	0	0	0	
4	ANDHRA PRADESH	Groundnut	Autumn	1997	26	1	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	.
3402	WEST BENGAL	Rice	Autumn	2014	26	1	0	0	1	0	
3403	WEST BENGAL	Rice	Summer	2014	28	1	0	0	0	1	
3404	WEST BENGAL	Rice	Winter	2014	20	1	0	0	0	0	
3405	WEST BENGAL	Sesamum	Summer	2014	28	1	0	0	0	1	
3406	WEST BENGAL	Wheat	Winter	2014	20	0	1	0	0	0	

3407 rows × 14 columns

2. Feature selection

Splitting data into Input and Output for feature selection

In [42]:

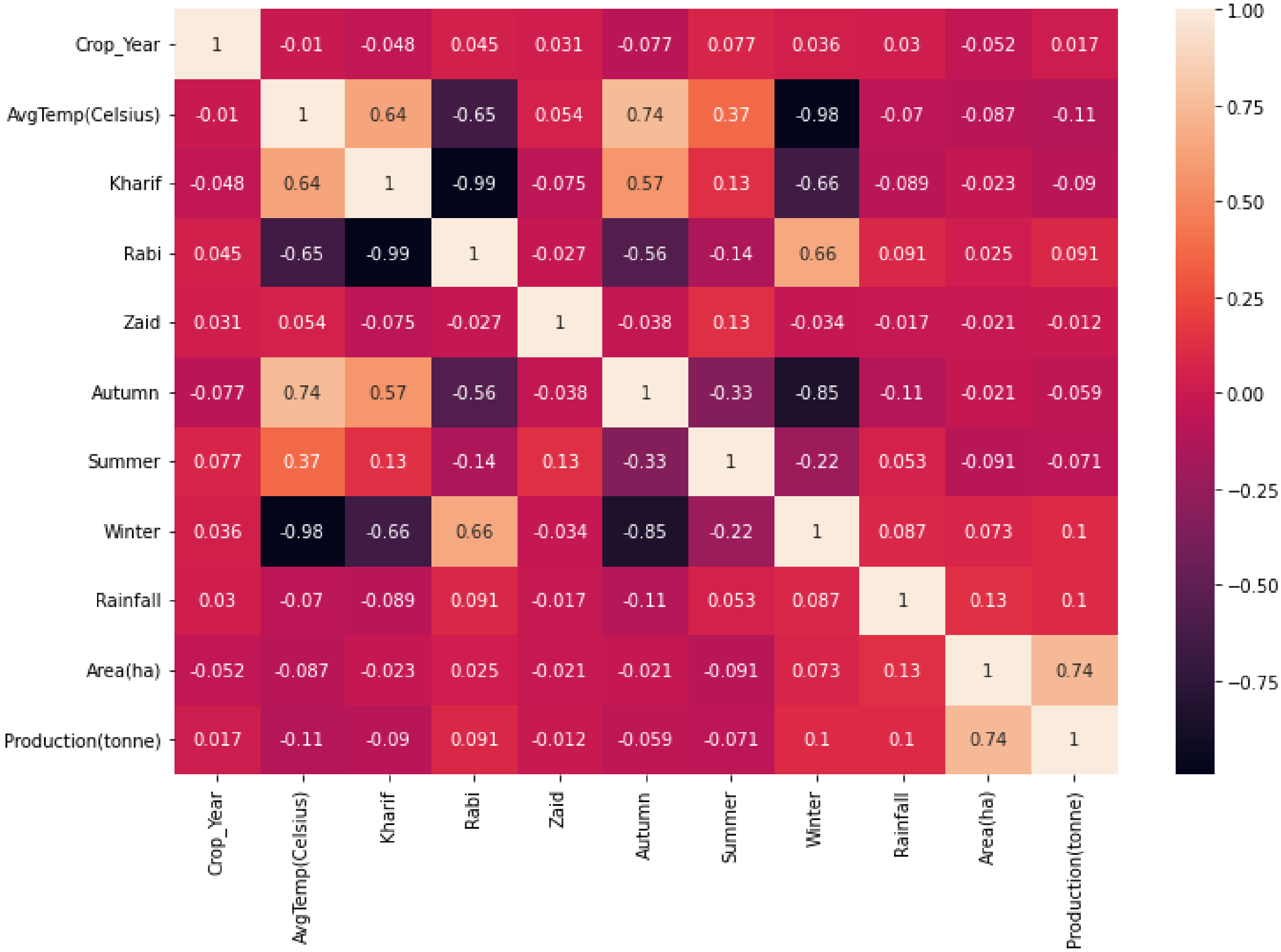
```
array = df_merged_states_refined.values
X = array[:, 4:13]
y = array[:, 13]
```

In [43]:

```
plt.subplots(figsize=(12,8))
sns.heatmap(df_merged_states_refined.corr(), annot = True)
```

Out[43]:

```
<AxesSubplot:>
```



Analysis:

From the above correlation matrix we can see that Area, Rainfall, Temperature and Seasons have good correlations.

To further check for the best features, I decided to use a FEATURE SELECTION technique

Feature selection technique

The feature selection technique I decided on using is ANOVA, this is becuase my features contain categorical input in form of one hot encoded values and the target variable is numerical

In [44]:

```
fs = SelectKBest(score_func=f_classif, k = 5)
result = fs.fit_transform(X, y)
```

In [45]:

```
result
```

Out[45]:

```
array([[26, 0, 0, 10693.1, 59300.0],
       [26, 0, 0, 11665.2, 368100.0],
       [26, 0, 0, 10693.1, 57200.0],
       ...,
       [20, 0, 1, 22705.2, 4008662.0],
       [28, 0, 0, 21443.800000000003, 222239.0],
       [20, 0, 1, 22705.2, 334640.0]], dtype=object)
```

The above result tells me that Area, rainfall, Avg(Temperature) and Two of my one hot encoded values are the top 4 best features for predicting how much will be produced in tonnes.

To find what the other two one hot encoded values are, I will have to fit the input and output values but without tranforming the result

In [46]:
```
fs2 = SelectKBest(score_func=f_classif, k = 5)
result2 = fs2.fit(X, y)
```

In [47]:
```
print(result2.scores_)
```

```
[   1.51676899    0.88205735    0.87762781           inf    1.48923789
    0.86812919    1.57651877    3.70229552 237.54397622]
```

Form the above we can see each feature score, Area has a strong correlation, Rainfall next, the seasons in which the crop was planted and temperature have good correlations

3. Models traninig and testing

Model

The models which i have chosen to work with

- Linear regression
- Decision Tree regressor
- Randomforest regressor

Evaluation

The evaluation techniques which I have chosen to work with is

- Mean absolute error
  MAE gets how much off a models prediciton is from the actual value(Absolute Error) and gives a mean of the errors(MAE)
  Therefore models with their MAE closest to 0 will be the best

- R^2
  Is how accurate a model is at predicting

In [48]:
```
df_merged_states_refined
```

Out[48]:

	State_Name	Crop	Season	Crop_Year	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winte
0	ANDHRA PRADESH	Bajra	Autumn	1997	26	1	0	0	1	0	
1	ANDHRA PRADESH	Cotton(lint)	Autumn	1997	26	1	0	0	1	0	
2	ANDHRA PRADESH	Dry chillies	Autumn	1997	26	1	0	0	1	0	
3	ANDHRA PRADESH	Gram	Winter	1997	20	0	1	0	0	0	
4	ANDHRA PRADESH	Groundnut	Autumn	1997	26	1	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	.

	State_Name	Crop	Season	Crop_Year	AvgTemp(Celsius)	Kharif	Rabi	Zaid	Autumn	Summer	Winte
3402	WEST BENGAL	Rice	Autumn	2014	26	1	0	0	1	0	
3403	WEST BENGAL	Rice	Summer	2014	28	1	0	0	0	1	
3404	WEST BENGAL	Rice	Winter	2014	20	1	0	0	0	0	
3405	WEST BENGAL	Sesamum	Summer	2014	28	1	0	0	0	1	
3406	WEST BENGAL	Wheat	Winter	2014	20	0	1	0	0	0	

3407 rows × 14 columns

3a. Choosing and Splitting features

In [49]:

```
x_model = df_merged_states_refined.iloc[:, 4:13]
y_model = df_merged_states_refined["Production(tonne)"]
```

In [50]:

```
x_model.drop("Zaid", axis = "columns", inplace = True)
x_model.drop("Kharif", axis = "columns", inplace = True)
x_model.drop("Rabi", axis = "columns", inplace = True)
```

In [51]:

```
# I decided to drop winter because it is good practise to drop one one hot encoded feature
x_model.drop("Winter", axis = "columns", inplace = True)
```

In [52]:

```
X_train, X_test, y_train, y_test = train_test_split(x_model, y_model, test_size=0.2)
```

3b. Modelling application

3bi. LINEAR REGRESSION

In [53]:

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Out[53]:

```
LinearRegression()
```

In [54]:

```
predictionte = lr.predict(X_test)
lrte_prediction = lr.predict(X_test)
lrtr_prediction = lr.predict(X_train)
predictiontr = lr.predict(X_train)
```

EVALUATION

In [55]:

```
print('R^2 test: %.3f R^2 train: %.3f' %
    (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

```
R^2 test: 0.665 R^2 train: 0.512
```

In [56]:

```
mean_absolute_error(y_test, predictionte)
```

Out[56]: 797148.2842928547

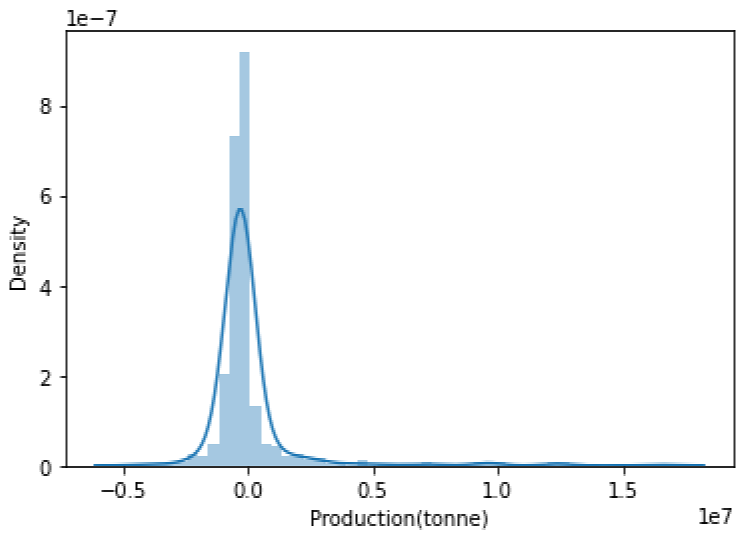
In [57]:
```
mean_absolute_error(y_train, predictiontr)
```

Out[57]: 890693.7619683887

Comparing Range of Predicted value to the actual for predicted test data

In [58]:
```
sns.distplot(y_test-lrte_prediction)
```

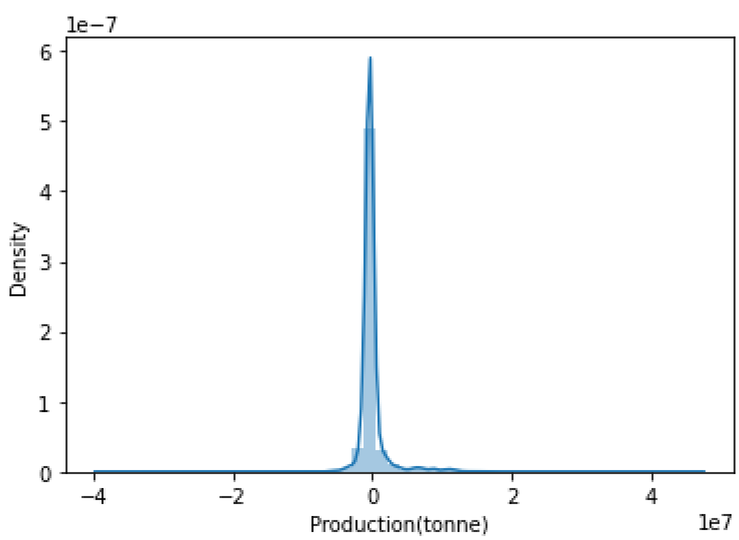
Out[58]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



Comparing Range of Predicted value to the actual for predicted train data

In [59]:
```
sns.distplot(y_train-lrtr_prediction)
```

Out[59]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



Conclusion after model prediction:

From the above evaluation, we can see that the Linear regression model(Without hyperparameter tunning, cross validation and feature scaling) has a higher test result score (R^2), but a low training data prediction. The observation is better seen with the Mean aboslute score.

The MAE for predicition on training is higher than the model prediciton of test data, due to this I can conclude that

The model has HIGH BIAS and LOW VARIANCE

3bii. DECISION TREE REGRESSOR

Hypothesis: Model will have a high bias because it is also a simple model like linear regression

In [60]:
```
drg = DecisionTreeRegressor()
drg.fit(X_train, y_train)
```

Out[60]:
```
DecisionTreeRegressor()
```

In [61]:
```
predictionte = drg.predict(X_test)
dt_prediction = drg.predict(X_test)
dtr_prediction = drg.predict(X_train)
predictiontr = drg.predict(X_train)
```

EVALUATION

In [62]:
```
mean_absolute_error(y_test, predictionte)
```

Out[62]:
```
685523.8603225807
```

In [63]:
```
mean_absolute_error(y_train, predictiontr)
```

Out[63]:
```
568.5702752293578
```

In [64]:
```
print('R^2 test: %.3f R^2 train: %.3f' %
     (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

```
R^2 test: 0.385 R^2 train: 1.000
```

Conclusion after model prediction:

After running the model prediction, I noticed that the model training has a good test score(R^2) lower than train and low MAE for training. This model is better than the linear regression model at first glance.

An intial conclusion is this model is also HIGH VARIANCE and LOW BIAS

3b iii. RANDOM FOREST REGRESSOR

Hypothesis: Model will not have BIAS because it is a complex model

In [65]:
```
regr = RandomForestRegressor()
regr.fit(X_train, y_train)
```

Out[65]:
```
RandomForestRegressor()
```

In [66]:
```
predictionte = regr.predict(X_test)
predictiontr = regr.predict(X_train)
```

EVALUATION

In [67]:
```
mean_absolute_error(y_test, predictionte)
```

Out[67]: 632530.3294801075

In [68]:
```
mean_absolute_error(y_train, predictiontr)
```

Out[68]: 247601.3859994862

In [69]:
```
print('R^2 test: %.3f R^2 train: %.3f' %
     (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

```
R^2 test: 0.755 R^2 train: 0.946
```

Conclusion after model prediction:

After running the model, I noticed that the model evaluation scores for train dataset prediciton are good and the test data prediciton are better than the decision tree.
My hypothesis on the model [redacted] having a high variance is correct, this hypothesis was based on complex models have a tendency to overfit.

An intial conclusion is the model has HIGH VARIANCE and LOW BIAS.

4. Cross validation

After accessing my three models, I do not know which model is the best due to the values of all three models fluctuation with every new train and test dataset, therefore I have decided to apply CROSS VALIDATION to help determine which model is the best

Cross validation method selection:

- K Fold Cross validation
- Repeated K Fold Cross validation

I have decided to make use of Repeated K Fold Cross validation.

Reason:

I have chosen Repeated K Fold Cross validation because, with K Fold cross validation there is a chance the model will be biased and already since all my models are biased a solution to getting the actual best model is by using Repeated K Fold Cross Validation.
This repeats the K Fold Cross Validation process multiple times and reports the mean performance accross each folds and repeats.

E.g of Repeated K Fold Cross Validation:

If 10 fold cross validation is repeated 5 times 50 different sets would be used to estimate the model

CROSS VALIDATION LINEAR REGRESSION

In [70]:
```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(LinearRegression(), X_test, y_test, cv = cv, scoring='r2')
score.mean()
```

Out[70]: 0.6225486272877996

In [71]:

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(DecisionTreeRegressor(), X_test, y_test, cv = cv, scoring='r2')
score.mean()
```

Out[71]: 0.5800896286686361

In [72]:
```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(RandomForestRegressor(), X_test, y_test, cv = cv, scoring='r2')
score.mean()
```

Out[72]: 0.6608025752134635

From the above results, my best model is random forest with an average R^2 score of 0.75 which is 75%

4. Normalizing Data(Feature Scaling)

After cross validation, although I know my best model is random forest I have decided to try normalizing my feature for the linear regression model to see if i can make it better.

Intializing standard scalar

In [73]:
```
std = StandardScaler()
```

In [74]:
```
X_train_scaled = std.fit_transform(X_train)
```

In [75]:
```
X_test_scaled = std.transform(X_test)
```

LINEAR REGRESSION WITH SCALED INPUT

In [76]:
```
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
```

Out[76]: LinearRegression()

In [77]:
```
predictionte = lr.predict(X_test_scaled)
lrte_prediction = lr.predict(X_test_scaled)
lrtr_prediction = lr.predict(X_train_scaled)
predictiontr = lr.predict(X_train_scaled)
```

In [78]:
```
print('R^2 test: %.3f R^2 train: %.3f' %
     (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

```
R^2 test: 0.665 R^2 train: 0.512
```

In [79]:
```
mean_absolute_error(y_test, predictionte)
```

Out[79]: 797148.2842993237

In [80]:
```
mean_absolute_error(y_train, predictiontr)
```

890693.7619740115

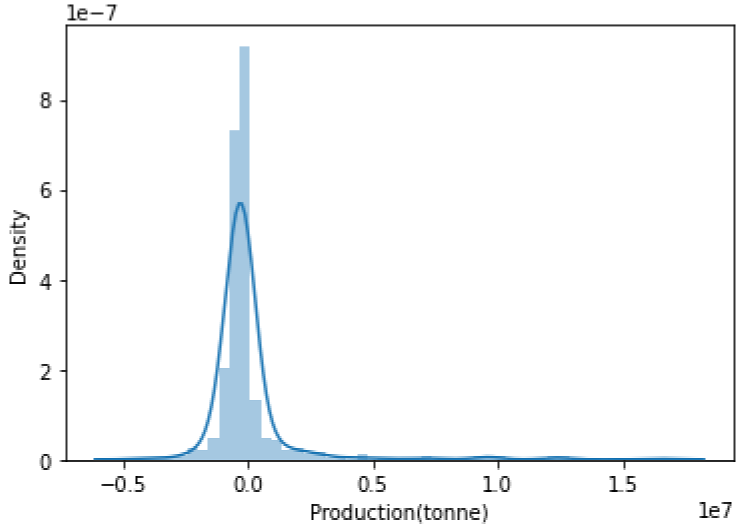
Out[80]:

Graphical representation of range of Prediciton value and actual value

In [81]:

```
sns.distplot(y_test-lrte_prediction)
```

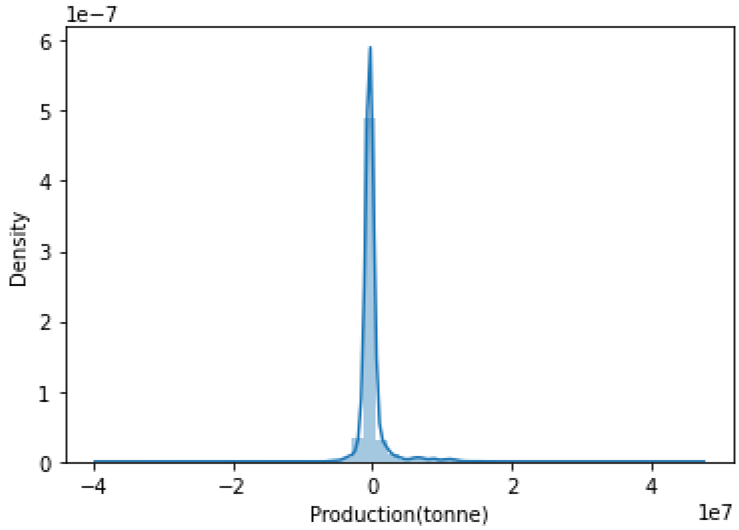
Out[81]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



In [82]:

```
sns.distplot(y_train-lrtr_prediction)
```

Out[82]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



Applying Repeated K fold cross validation to get accuracy

In [83]:

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(LinearRegression(), X_train, y_train, cv = cv, scoring='r2')
score.mean()
```

Out[83]: 0.45860702025183675

In [84]:

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(LinearRegression(), X_test, y_test, cv = cv, scoring='r2')
score.mean()
```

Out[84]: 0.6351328635366652

Conclusion:

From the result above, I noticed that even after scaling my features using standard scalar, the accuracy of my model remains the same and

The model remains BIASED (Underfit)

Note:

I will not be making use of other linear regression algorithms like Lasso and Ridge because these increase BIAS and decrease Variance

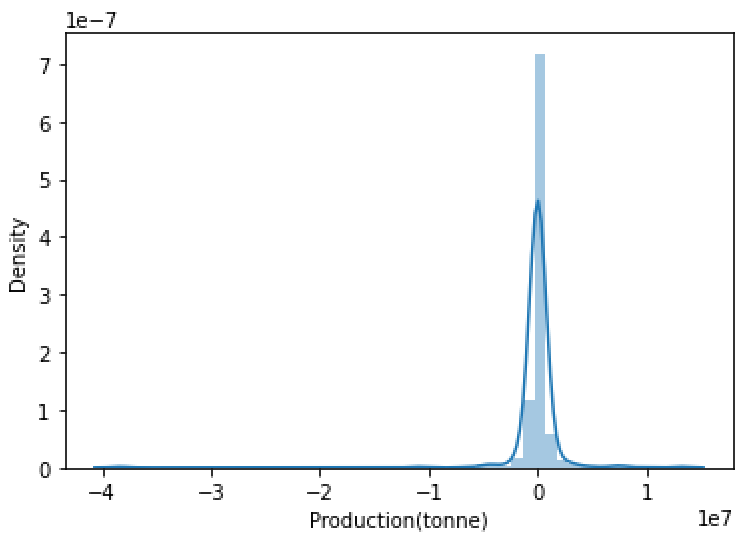
5. Grid Search

I decided to use Grid Search for tunning my models Decision Tree and Random Forest to see if by adding hyperparameters there would be any change to the model accuracy or not

In [85]:

```
sns.distplot(y_test-dt_prediction)
```

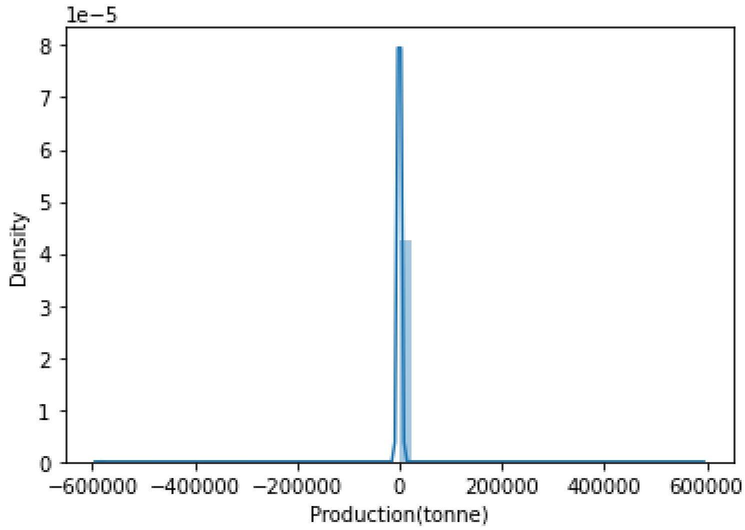
Out[85]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



In [86]:

```
sns.distplot(y_train-dtr_prediction)
```

Out[86]: <AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>



Form the graph above we can see that our Decision model (without Hyperparameter tunning or feature scaling) predicted values are almost the same range as our original data both for the test and train data.
Therefore hyperparameter tunning may be what is needed to make the model better

5a. Decision Tree

In [87]:
```
cv = RepeatedKFold(n_splits=5, n_repeats=5)
splitter_range =["best", "random"]
max_depth_range = np.arange(20, 40, 20)
max_features_range = ["auto", "log2", "sqrt", None]
max_leaf_nodes_range = np.arange(25,50, 25)

param_grid = dict(splitter = splitter_range, max_depth=max_depth_range, max_features = max

dt = DecisionTreeRegressor()

dt_grid = GridSearchCV(estimator = dt, scoring='neg_mean_absolute_error', param_grid=param
```

In [88]:
```
dt_grid.fit(X_train, y_train)
```

Out[88]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': array([20]),
                         'max_features': ['auto', 'log2', 'sqrt', None],
                         'max_leaf_nodes': array([25]),
                         'splitter': ['best', 'random']},
             scoring='neg_mean_absolute_error')
```

In [89]:
```
print("The best parameters are %s with a score of %0.2f" % (dt_grid.best_params_, dt_grid.
```

```
The best parameters are {'max_depth': 20, 'max_features': 'auto', 'max_leaf_nodes': 25, 's
plitter': 'best'} with a score of -742870.62
```

Testing the hyperparameter tunning

In [90]:
```
drg = DecisionTreeRegressor(max_depth=20, max_features = "auto", max_leaf_nodes=25, splitt
drg.fit(X_train, y_train)
```

Out[90]:
```
DecisionTreeRegressor(max_depth=20, max_features='auto', max_leaf_nodes=25,
                      splitter='random')
```

In [91]:
```
predictionte = drg.predict(X_test)
dt_prediction = drg.predict(X_test)
```

```
dtr_prediction = drg.predict(X_train)
predictiontr = drg.predict(X_train)
```

In [92]:

```
print('R^2 test: %.3f R^2 train: %.3f' %
    (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

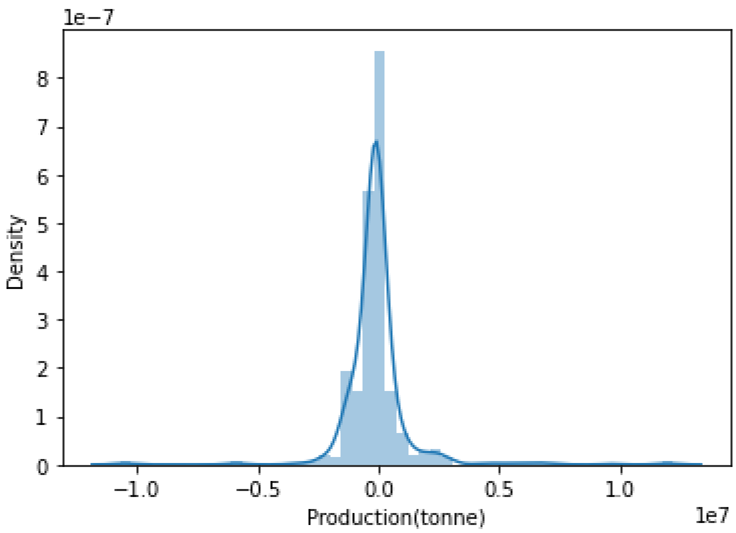
```
R^2 test: 0.812 R^2 train: 0.700
```

In [93]:

```
sns.distplot(y_test-dt_prediction)
```

Out[93]:

```
<AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>
```

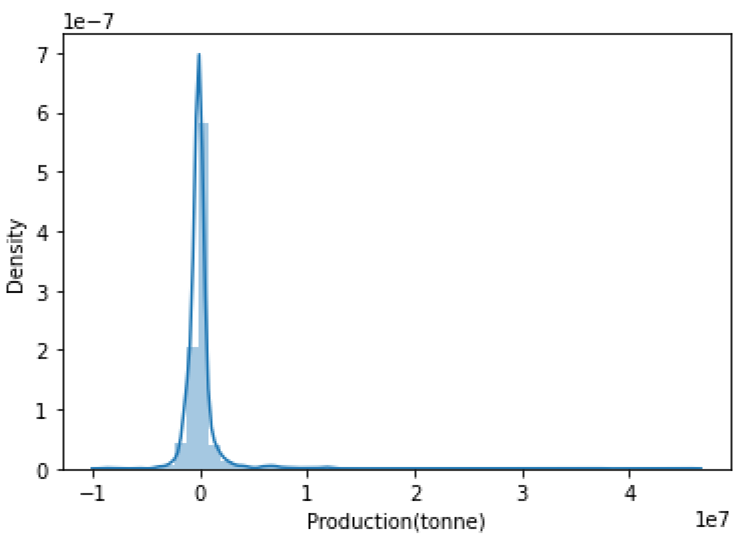


In [94]:

```
sns.distplot(y_train-dtr_prediction)
```

Out[94]:

```
<AxesSubplot:xlabel='Production(tonne)', ylabel='Density'>
```



Applying Repeated K Fold cross validation on Decision tree model with hyperparameters to get average accuracy

In [95]:

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(DecisionTreeRegressor(max_depth=20, max_features = "auto", max_lea
score.mean()
```

Out[95]:

```
0.6491759128422423
```

In [96]:

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(DecisionTreeRegressor(max_depth=20, max_features = "auto", max_lea
score.mean()
```

Out[96]:

```
0.6492889584669405
```

From the above result, after the introduction of hyperparameters from GridSearch the DecisionTree Model became high BIAS (Under fit) having its test prediction score higher than its train prediction with the introduction of hyperparameters

5b. Random Forest

To increase Variance and reduce bias I decided to also tune Random Forest, this is because as a complex model(ensemble algorithm) it should have more variance and lower bias, but from the first model creation without the addition of hyperparameters we can see that this is not so.

Therefore I am going to test if even after tunning the result remains the same.

Hyperparameters I need for raising variance:

- max_depth
- n_estimators
- Max_features
- Max_leaf_nodes

In [97]:

```
cv = RepeatedKFold(n_splits=5, n_repeats=5)
max_features_range = ["auto", "sqrt", "log2", np.arange(1,20,1)]
max_leaf_nodes_range = np.arange(25,50, 25)
n_estimators_range = np.arange(10,20,10)
# n_randomstate_range = np.arange(1, 10, 1)
max_depth_range = np.arange(20, 40, 20)

param_grid = dict(max_leaf_nodes = max_leaf_nodes_range , max_features = max_features_rang

rf = RandomForestRegressor()

#the scoring=neg_mean_absolute_error is done because performance in regression models is r
#Zero representing a model with perfect skill
#Good performance models are small negative values
rf_grid = GridSearchCV(estimator = rf, scoring='neg_mean_absolute_error', param_grid=param
```

In [98]:

```
rf_grid.fit(X_train, y_train)
```

Out[98]:

```
GridSearchCV(cv=RepeatedKFold(n_repeats=5, n_splits=5, random_state=None),
             estimator=RandomForestRegressor(),
             param_grid={'max_depth': array([20]),
                         'max_features': ['auto', 'sqrt', 'log2',
                                          array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
1, 12, 13, 14, 15, 16, 17,
       18, 19])],
                         'max_leaf_nodes': array([25]),
                         'n_estimators': array([10])},
             scoring='neg_mean_absolute_error')
```

In [99]:

```
print("The best parameters are %s with a score of %0.2f" % (rf_grid.best_params_, rf_grid.
```

```
The best parameters are {'max_depth': 20, 'max_features': 'auto', 'max_leaf_nodes': 25, 'n
_estimators': 10} with a score of -705012.39
```

Testing the hyperparameter tunning

In [121…

```
regr = RandomForestRegressor(max_depth=23, max_features="auto", max_leaf_nodes = 25, n_est
regr.fit(X_train, y_train)
```

Out[121…

```
RandomForestRegressor(max_depth=23, max_leaf_nodes=25, n_estimators=10)
```

In [122…

```
predictionte = regr.predict(X_test)
predictiontr = regr.predict(X_train)
```

In [123…

```
print('R^2 test: %.3f R^2 train: %.3f' %
    (r2_score(y_test, predictionte), r2_score(y_train, predictiontr)))
```

```
R^2 test: 0.727 R^2 train: 0.804
```

In [124…

```
mean_absolute_error(y_test, predictionte)
```

Out[124…

```
675004.6682126466
```

In [125…

```
mean_absolute_error(y_train, predictiontr)
```

Out[125…

```
595804.426111369
```

Below I have tried using repeated k fold cross validation with my optmized model

In [126…

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(RandomForestRegressor(max_depth=23, max_features="auto", max_leaf_
score.mean()
```

Out[126…

```
0.6251387269226133
```

In [127…

```
cv = RepeatedKFold(n_splits=10, n_repeats=5)

score = cross_val_score(RandomForestRegressor(max_depth=23, max_features="auto", max_leaf_
score.mean()
```

Out[127…

```
0.6055258139840667
```

Conclusion:

With the Repeated cross validation on Random forest with grid search applied, I noticed that the random forest model after tunning is the best model overall for my predicitive problem

From the above results I have been able to reach a good spot whereby predicition on training dataset is not too large and the predicition on test dataset is good enough. With the help of Grid Search and general model understanding I was able to acheive this.

The main features which helped me stabilize my model are:

- max_depth (important)

- n_estimators
- Max_features
- Max_leaf_nodes