In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

In [6]:
```
df=pd.read_csv('C:\\Users\\RISHI\\OneDrive\\Documents\\archive (2)\\global_
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 19 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Year    144 non-null    int64  
 1   Jan     144 non-null    float64
 2   Feb     144 non-null    float64
 3   Mar     144 non-null    float64
 4   Apr     144 non-null    float64
 5   May     144 non-null    float64
 6   Jun     143 non-null    float64
 7   Jul     143 non-null    float64
 8   Aug     143 non-null    float64
 9   Sep     143 non-null    float64
 10  Oct     143 non-null    float64
 11  Nov     143 non-null    float64
 12  Dec     143 non-null    float64
 13  J-D     143 non-null    float64
```

In [7]:
```
df
```

Out[7]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	J-D
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.21	-0.18	-0.11	-0.15	-0.24	-0.22	-0.18	-0.17
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.19	0.00	-0.04	-0.16	-0.22	-0.19	-0.08	-0.09
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.22	-0.17	-0.08	-0.15	-0.24	-0.17	-0.36	-0.11
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.08	-0.08	-0.14	-0.23	-0.12	-0.24	-0.11	-0.18
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.35	-0.31	-0.28	-0.28	-0.25	-0.34	-0.31	-0.29
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
139	2019	0.93	0.95	1.17	1.02	0.85	0.91	0.94	0.95	0.93	1.01	1.00	1.09	0.98
140	2020	1.18	1.25	1.17	1.13	1.02	0.92	0.90	0.88	0.99	0.89	1.11	0.81	1.02
141	2021	0.81	0.64	0.89	0.76	0.79	0.84	0.92	0.82	0.93	1.00	0.94	0.86	0.85
142	2022	0.91	0.90	1.05	0.84	0.84	0.93	0.94	0.96	0.90	0.97	0.73	0.80	0.90
143	2023	0.87	0.98	1.21	1.00	0.94	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

144 rows × 19 columns

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Year    0
Jan     0
Feb     0
Mar     0
Apr     0
May     0
Jun     1
Jul     1
Aug     1
Sep     1
Oct     1
Nov     1
Dec     1
J-D     1
D-N     2
DJF     1
MAM     0
JJA     1
SON     1
dtype: int64
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['Year', 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Se
p',
       'Oct', 'Nov', 'Dec', 'J-D', 'D-N', 'DJF', 'MAM', 'JJA', 'SON'],
      dtype='object')
```

In [10]:
```
I = ['Year', 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
       'Oct', 'Nov', 'Dec', 'J-D', 'D-N', 'DJF', 'MAM', 'JJA', 'SON'],

for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

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

Out[7]:
```
Year    0
Jan     0
Feb     0
Mar     0
Apr     0
May     0
Jun     0
Jul     0
Aug     0
Sep     0
Oct     0
Nov     0
Dec     0
J-D     0
D-N     0
DJF     0
MAM     0
JJA     0
SON     0
dtype: int64
```

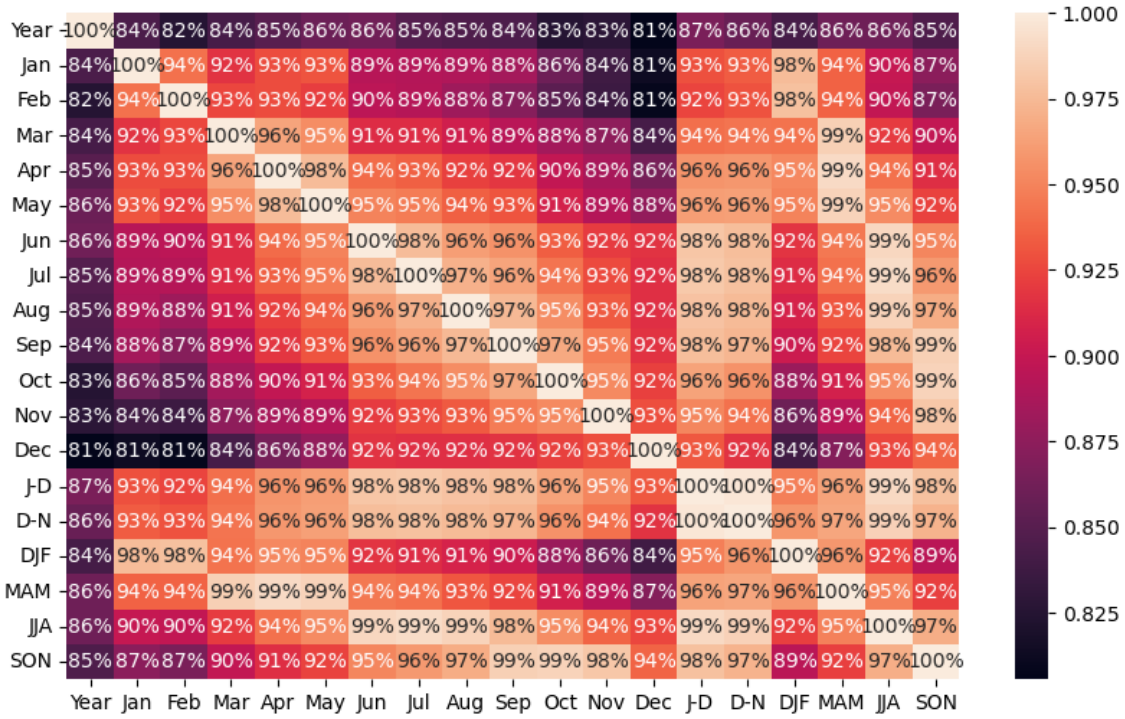
In [8]:
```
df.corr()
```

Out[8]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	A
Year	1.000000	0.844005	0.824928	0.841728	0.849647	0.858926	0.861303	0.848543	0.8453
Jan	0.844005	1.000000	0.942181	0.916700	0.927742	0.930311	0.892972	0.891407	0.8904
Feb	0.824928	0.942181	1.000000	0.925318	0.927608	0.921069	0.895207	0.892856	0.882(
Mar	0.841728	0.916700	0.925318	1.000000	0.964790	0.954391	0.914032	0.914093	0.9057
Apr	0.849647	0.927742	0.927608	0.964790	1.000000	0.975763	0.939386	0.934824	0.9231
May	0.858926	0.930311	0.921069	0.954391	0.975763	1.000000	0.949133	0.946600	0.9351
Jun	0.861303	0.892972	0.895207	0.914032	0.939386	0.949133	1.000000	0.976237	0.9618
Jul	0.848543	0.891407	0.892856	0.914093	0.934824	0.946600	0.976237	1.000000	0.9708
Aug	0.845397	0.890437	0.882066	0.905762	0.923145	0.935187	0.961882	0.970877	1.0000
Sep	0.844394	0.884906	0.871039	0.894065	0.918603	0.930332	0.956123	0.964998	0.9747
Oct	0.825105	0.860590	0.851688	0.884474	0.898497	0.907893	0.934676	0.943752	0.9542
Nov	0.827790	0.837575	0.836302	0.874997	0.886119	0.891474	0.920557	0.926968	0.9328
Dec	0.805555	0.812417	0.808354	0.841845	0.864133	0.877187	0.916735	0.915560	0.9151
J-D	0.865840	0.928926	0.924221	0.941148	0.956740	0.961689	0.980281	0.981821	0.9788
D-N	0.857840	0.933589	0.929644	0.943622	0.958267	0.962081	0.976883	0.978054	0.9755
DJF	0.840995	0.975914	0.978104	0.942487	0.951741	0.949376	0.917872	0.914130	0.9087
MAM	0.860482	0.935461	0.935630	0.985741	0.991609	0.987255	0.944621	0.942184	0.9314
JJA	0.860856	0.899917	0.898698	0.920302	0.941567	0.953090	0.989468	0.992105	0.9877
SON	0.845601	0.873760	0.865974	0.898435	0.914539	0.923606	0.951098	0.959404	0.9682

In [9]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, fmt='.0%')
```

Out[9]: <Axes: >



In [10]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.037931034482758634
Root mean square error:  8.038049940546974e-05
R2 square:  0.9733370581298103
Mean absolute Error:  2.051499067242137e-18
Root mean square error:  7.135770682417736e-37
R2 square:  1.0
```

In [11]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.022067189728463468
Root mean square error:  6.361710998885262e-05
R2 square:  0.9956972348933649
Mean absolute Error:  0.0013855265147008373
Root mean square error:  5.418816879444163e-35
R2 square:  0.9999720004687392
```

In [12]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04448275862068966
Root mean square error:  1.070154577883469e-06
R2 square:  0.9692612795382934
Mean absolute Error:  0.0
Root mean square error:  0.0
R2 square:  1.0
```

In [13]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.019209404388714754
Root mean square error:  2.5982705500514823e-05
R2 square:  0.9956758714258298
Mean absolute Error:  0.007347114624505921
Root mean square error:  1.6400899248547113e-07
R2 square:  0.9991837002863515
```

In [14]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.026386952820197488
Root mean square error:  0.00013964478189774983
R2 square:  0.985962695946292
Mean absolute Error:  0.01591085199385222
Root mean square error:  3.982577064688006e-09
R2 square:  0.996666208425852
```

In [15]:
```
df['new_feature1'] = abs(df['JJA'] + df['JJA'])
df['new_feature1']
```

Out[15]:
```
0      0.340000
1      0.160000
2      0.320000
3      0.200000
4      0.640000
         ...
139    1.860000
140    1.800000
141    1.720000
142    1.880000
143    0.095385
Name: new_feature1, Length: 144, dtype: float64
```

In [16]:
```
df
```

Out[16]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 20 columns

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
Year            0
Jan             0
Feb             0
Mar             0
Apr             0
May             0
Jun             0
Jul             0
Aug             0
Sep             0
Oct             0
Nov             0
Dec             0
J-D             0
D-N             0
DJF             0
MAM             0
JJA             0
SON             0
new_feature1    0
dtype: int64
```

In [18]:
```
df["Jan"].unique()
```

Out[18]:
```
array([-0.19, -0.2 ,  0.16, -0.3 , -0.13, -0.59, -0.44, -0.72, -0.34,
       -0.09, -0.42, -0.33, -0.29, -0.81, -0.53, -0.43, -0.22, -0.15,
       -0.02, -0.17, -0.36, -0.21, -0.18, -0.23, -0.64, -0.35, -0.28,
       -0.63, -0.27, -0.4 ,  0.05, -0.12, -0.57, -0.47, -0.24, -0.05,
       -0.39,  0.2 , -0.45, -0.1 , -0.07,  0.08,  0.  ,  0.18,  0.29,
       -0.01,  0.36,  0.1 ,  0.15, -0.06,  0.07, -0.26,  0.11,  0.13,
        0.39, -0.03, -0.08, -0.11,  0.06,  0.3 ,  0.53,  0.31,  0.22,
        0.26,  0.32,  0.57,  0.41,  0.43,  0.48,  0.35,  0.52,  0.24,
        0.58,  0.49,  0.25,  0.46,  0.78,  0.75,  0.74,  0.56,  1.02,
        0.65,  0.71,  0.76,  0.86,  1.17,  0.82,  0.93,  1.18,  0.81,
        0.91,  0.87])
```

In [19]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.05413793103448275
Root mean square error:  0.0007420927467300833
R2 square:  0.9660215043677816
Mean absolute Error:  2.172175482962263e-18
Root mean square error:  5.242607031980377e-37
R2 square:  1.0
```

In [20]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.022618578872034365
Root mean square error:  6.291667531019526e-07
R2 square:  0.9921102501184156
Mean absolute Error:  0.001242029601023142
Root mean square error:  2.8187409159719186e-34
R2 square:  0.9999817031666471
```

In [21]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.043448275862068966
Root mean square error:  1.189060642092747e-05
R2 square:  0.9728782674673461
Mean absolute Error:  2.051499067242137e-18
Root mean square error:  7.135770682417736e-37
R2 square:  1.0
```

In [22]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.019368820834338028
Root mean square error:  0.00016216336622666436
R2 square:  0.9957853274651043
Mean absolute Error:  0.009392982669504397
Root mean square error:  3.376476085418511e-08
R2 square:  0.9981031409445172
```

In [23]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.028504534394406745
Root mean square error:  3.532036756216614e-07
R2 square:  0.988925146389184
Mean absolute Error:  0.01723238809161146
Root mean square error:  5.917243827004842e-07
R2 square:  0.9963296994181349
```

In [24]:
```
df.drop('new_feature1',axis=1,inplace=True)
```

In [25]:
```
df
```

Out[25]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [26]:
```
df['new_feature1'] = abs(df['SON'] - df['JJA'])
df['new_feature1']
```

Out[26]:
```
0      0.030000
1      0.110000
2      0.030000
3      0.100000
4      0.030000
         ...
139    0.050000
140    0.090000
141    0.100000
142    0.070000
143    0.025175
Name: new_feature1, Length: 144, dtype: float64
```

In [27]:
```
df
```

Out[27]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 20 columns

In [28]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024827586206896558
Root mean square error:  1.712247324613559e-05
R2 square:  0.9935069259928437
Mean absolute Error:  0.0
Root mean square error:  0.0
R2 square:  1.0
```

In [29]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.01880544371492275
Root mean square error:  0.00010258947011072299
R2 square:  0.9953825160229722
Mean absolute Error:  0.0012598043977657559
Root mean square error:  1.526274451119006e-34
R2 square:  0.9999786879753374
```

In [30]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.052068965517241377
Root mean square error:  1.0701545778834721e-06
R2 square:  0.9681328361722648
Mean absolute Error:  1.4481169886415085e-18
Root mean square error:  2.0970428127921508e-36
R2 square:  1.0
```

In [31]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.015664649143959514
Root mean square error:  3.897458594389533e-08
R2 square:  0.995126658424635
Mean absolute Error:  0.007800826999087856
Root mean square error:  1.865063317187054e-09
R2 square:  0.9992066528860138
```

In [32]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.023895015605767577
Root mean square error:  2.579460638516911e-05
R2 square:  0.9905152245653924
Mean absolute Error:  0.018328030314579375
Root mean square error:  2.7688165375251823e-06
R2 square:  0.9960647219651537
```

In [33]:
```
df.drop('new_feature1',axis=1,inplace=True)
```

In [34]:
```
df['Polynomial_Feature'] = df['SON'] * df['JJA']
df
```

Out[34]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 20 columns

In [35]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.032758620689655175
Root mean square error:  5.826397146254456e-06
R2 square:  0.9791267879384206
Mean absolute Error:  2.172175482962263e-18
Root mean square error:  5.242607031980377e-37
R2 square:  1.0
```

In [36]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024947982169186998
Root mean square error:  3.903880465947438e-05
R2 square:  0.9930404708457344
Mean absolute Error:  0.0013207774949190014
Root mean square error:  2.658987029469628e-34
R2 square:  0.9999769809556033
```

In [37]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.028965517241379315
Root mean square error:  4.75624256837099e-05
R2 square:  0.9642002168857121
Mean absolute Error:  1.3274405729213829e-18
Root mean square error:  1.762098474637849e-36
R2 square:  1.0
```

In [38]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.021927296841089898
Root mean square error:  3.327094784455149e-05
R2 square:  0.9943584290720632
Mean absolute Error:  0.00841905138339913
Root mean square error:  3.97356014155175e-07
R2 square:  0.9982445895873414
```

In [39]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024059928260771656
Root mean square error:  2.3269050039942357e-05
R2 square:  0.9918315664522391
Mean absolute Error:  0.018065295488426506
Root mean square error:  2.8724663078811867e-07
R2 square:  0.9960697044101574
```

In [40]:
```
df.drop('Polynomial_Feature',axis=1,inplace=True)
df
```

Out[40]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [41]:
```
df['Polynomial_Feature'] = df['JJA'] / df['SON']
df
```

Out[41]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 20 columns

In [42]:
```
df['Polynomial_Feature'].unique()
```

Out[42]:
```
array([  0.85      ,   0.42105263,   0.84210526,   0.5       ,
         1.10344828,   1.44      ,   1.03703704,   1.        ,
         4.66666667,   0.46428571,   0.85714286,   0.7826087 ,
         1.125     ,   1.05      ,   1.15384615,   1.46153846,
         3.        ,   0.61538462,   0.74193548, -19.        ,
        10.        ,   0.65217391,   0.93548387,   0.87234043,
         1.35135135,   1.47058824,   0.72413793,   1.08823529,
         0.88636364,   1.16216216,   0.8       ,   1.5       ,
         0.78846154,   1.31034483,   1.90909091,   0.97435897,
         0.88235294,   2.75      ,   1.13793103,   1.24      ,
         1.91666667,   1.07142857,   1.9375    ,   1.03448276,
         2.6       ,   2.        ,   3.83333333,   1.6875    ,
         2.23529412,   6.33333333,   1.66666667,   0.92857143,
         1.35294118,  -0.22222222,  -1.25      ,   6.        ,
         0.71428571,   0.33333333,   0.125     ,   0.77272727,
         0.66666667,   2.57142857,   4.        ,   0.9       ,
         1.8       ,   0.45454545,   0.        ,  -1.        ,
        18.        ,   0.57142857,   0.89473684,   1.83333333,
        -3.        ,  -0.28571429,        -inf,   0.6875    ,
         0.37037037,   0.88888889,   0.1       ,  -0.66666667,
         1.33333333,   1.75      ,  -0.14285714,   0.6       ,
         0.91666667,   3.14285714,  -0.5       ,   0.42307692,
         0.95238095,   1.88235294,   0.53333333,   0.89285714,
         1.2       ,   1.3       ,   1.03125   ,   1.27586207,
         1.02631579,   1.53846154,   0.82051282,   1.04761905,
         1.32142857,   0.72881356,   1.64285714,   0.97222222,
         1.25      ,   0.9137931 ,   0.94915254,   0.9047619 ,
         0.63934426,   0.84931507,   0.92753623,   1.01694915,
         0.78787879,   0.95833333,   1.15      ,   0.81818182,
         0.88461538,   0.78      ,   0.97802198,   0.94117647,
         0.89772727,   0.94897959,   0.90909091,   0.89583333,
         1.08045977,   0.65451056])
```

In [43]:
```
column_name = 'Polynomial_Feature'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [44]:
```
df['Polynomial_Feature'].unique()
```

Out[44]:
```
array([  0.85      ,   0.42105263,   0.84210526,   0.5       ,
         1.10344828,   1.44      ,   1.03703704,   1.        ,
         4.66666667,   0.46428571,   0.85714286,   0.7826087 ,
         1.125     ,   1.05      ,   1.15384615,   1.46153846,
         3.        ,   0.61538462,   0.74193548, -19.        ,
        10.        ,   0.65217391,   0.93548387,   0.87234043,
         1.35135135,   1.47058824,   0.72413793,   1.08823529,
         0.88636364,   1.16216216,   0.8       ,   1.5       ,
         0.78846154,   1.31034483,   1.90909091,   0.97435897,
         0.88235294,   2.75      ,   1.13793103,   1.24      ,
         1.91666667,   1.07142857,   1.9375    ,   1.03448276,
         2.6       ,   2.        ,   3.83333333,   1.6875    ,
         2.23529412,   6.33333333,   1.66666667,   0.92857143,
         1.35294118,  -0.22222222,  -1.25      ,   6.        ,
         0.71428571,   0.33333333,   0.125     ,   0.77272727,
         0.66666667,   2.57142857,   4.        ,   0.9       ,
         1.8       ,   0.45454545,   0.        ,  -1.        ,
        18.        ,   0.57142857,   0.89473684,   1.83333333,
        -3.        ,  -0.28571429,          nan,   0.6875    ,
         0.37037037,   0.88888889,   0.1       ,  -0.66666667,
         1.33333333,   1.75      ,  -0.14285714,   0.6       ,
         0.91666667,   3.14285714,  -0.5       ,   0.42307692,
         0.95238095,   1.88235294,   0.53333333,   0.89285714,
         1.2       ,   1.3       ,   1.03125   ,   1.27586207,
         1.02631579,   1.53846154,   0.82051282,   1.04761905,
         1.32142857,   0.72881356,   1.64285714,   0.97222222,
         1.25      ,   0.9137931 ,   0.94915254,   0.9047619 ,
         0.63934426,   0.84931507,   0.92753623,   1.01694915,
         0.78787879,   0.95833333,   1.15      ,   0.81818182,
         0.88461538,   0.78      ,   0.97802198,   0.94117647,
         0.89772727,   0.94897959,   0.90909091,   0.89583333,
         1.08045977,   0.65451056])
```

In [45]:
```
I = ['Polynomial_Feature'],

for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [46]:
```
df.isnull().sum()
```

Out[46]:
```
Year                0
Jan                 0
Feb                 0
Mar                 0
Apr                 0
May                 0
Jun                 0
Jul                 0
Aug                 0
Sep                 0
Oct                 0
Nov                 0
Dec                 0
J-D                 0
D-N                 0
DJF                 0
MAM                 0
JJA                 0
SON                 0
Polynomial_Feature  0
dtype: int64
```

In [47]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04206896551724139
Root mean square error:  1.7122473246135554e-05
R2 square:  0.9797361702019927
Mean absolute Error:  1.4481169886415085e-18
Root mean square error:  2.0970428127921508e-36
R2 square:  1.0
```

In [48]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024626299133048014
Root mean square error:  2.3932128000692603e-06
R2 square:  0.9883257438447663
Mean absolute Error:  0.0012175532651951587
Root mean square error:  1.734176586230978e-34
R2 square:  0.9999796514543299
```

In [49]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.05
Root mean square error:  0.00014565992865636147
R2 square:  0.9618557454829065
Mean absolute Error:  1.4481169886415085e-18
Root mean square error:  2.0970428127921508e-36
R2 square:  1.0
```

In [50]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.015392741741017598
Root mean square error:  4.5881908589733764e-06
R2 square:  0.9956663564327752
Mean absolute Error:  0.008218996655518385
Root mean square error:  1.6642629097739908e-07
R2 square:  0.9990565621322579
```

In [51]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.027692340201337227
Root mean square error:  0.00011452504341629993
R2 square:  0.9897420765662026
Mean absolute Error:  0.018524180232020362
Root mean square error:  5.186882424329348e-06
R2 square:  0.9959751407730554
```

In [52]:
```
df.drop('Polynomial_Feature',axis=1,inplace=True)
```

In [53]:
```
df['log_SON'] = np.log10(df['SON'])
df['log_JJA'] = np.log10(df['JJA'])
```

```
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: divide by zero encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: divide by zero encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [54]:
```
df
```

Out[54]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [55]:
```
df.isnull().sum()
```

Out[55]:
```
Year        0
Jan         0
Feb         0
Mar         0
Apr         0
May         0
Jun         0
Jul         0
Aug         0
Sep         0
Oct         0
Nov         0
Dec         0
J-D         0
D-N         0
DJF         0
MAM         0
JJA         0
SON         0
log_SON    76
log_JJA    79
dtype: int64
```

In [56]:
```
I = [ 'log_JJA','log_SON'],
for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [57]:
```
df.isnull().sum()
```

Out[57]:
```
Year       0
Jan        0
Feb        0
Mar        0
Apr        0
May        0
Jun        0
Jul        0
Aug        0
Sep        0
Oct        0
Nov        0
Dec        0
J-D        0
D-N        0
DJF        0
MAM        0
JJA        0
SON        0
log_SON    0
log_JJA    0
dtype: int64
```

In [58]:
```
df['log_JJA'].unique()
```

Out[58]:
```
array([       -inf, -1.        , -0.76955108, -2.        , -1.69897   ,
       -1.22184875, -0.95860731, -1.39794001, -1.15490196, -0.92081875,
       -0.65757732, -0.69897   , -0.49485002, -1.09691001, -0.60205999,
       -0.85387196, -0.88605665, -0.48148606, -0.43179828, -0.55284197,
       -0.40893539, -0.33724217, -0.35654732, -0.36653154, -0.16115091,
       -0.45593196, -0.39794001, -0.27572413, -0.25181197, -0.24412514,
       -0.20760831, -0.19382003, -0.22184875, -0.28399666, -0.18045606,
       -0.20065945, -0.1079054 , -0.05060999, -0.09691001, -0.10237291,
       -0.03151705, -0.04575749, -0.06550155, -0.02687215, -1.32155166])
```

In [59]:
```
df['log_SON'].unique()
```

Out[59]:
```
array([       -inf, -2.        , -1.04575749, -1.09691001, -0.85387196,
       -0.69897   , -1.52287875, -0.79588002, -0.65757732, -0.82390874,
       -1.39794001, -1.22184875, -1.69897   , -1.        , -1.15490196,
       -0.58502665, -0.67778071, -0.76955108, -0.55284197, -0.49485002,
       -0.537602  , -0.4202164 , -0.46852108, -0.88605665, -0.40893539,
       -0.37675071, -0.22914799, -0.4436975 , -0.23657201, -0.20065945,
       -0.21467016, -0.13667714, -0.16115091, -0.18045606, -0.1426675 ,
       -0.22184875, -0.11350927, -0.1079054 ,  0.        , -0.04095861,
       -0.07058107, -0.05551733, -0.00877392, -0.00436481, -0.01772877,
       -0.06048075, -1.13746832])
```

In [60]:
```
column_name = 'log_SON'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [61]:
```
column_name = 'log_JJA'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [62]:
```
df['log_SON'].unique()
```

Out[62]:
```
array([        nan, -2.        , -1.04575749, -1.09691001, -0.85387196,
       -0.69897   , -1.52287875, -0.79588002, -0.65757732, -0.82390874,
       -1.39794001, -1.22184875, -1.69897   , -1.        , -1.15490196,
       -0.58502665, -0.67778071, -0.76955108, -0.55284197, -0.49485002,
       -0.537602  , -0.4202164 , -0.46852108, -0.88605665, -0.40893539,
       -0.37675071, -0.22914799, -0.4436975 , -0.23657201, -0.20065945,
       -0.21467016, -0.13667714, -0.16115091, -0.18045606, -0.1426675 ,
       -0.22184875, -0.11350927, -0.1079054 ,  0.        , -0.04095861,
       -0.07058107, -0.05551733, -0.00877392, -0.00436481, -0.01772877,
       -0.06048075, -1.13746832])
```

In [63]:
```
df['log_JJA'].unique()
```

Out[63]:
```
array([        nan, -1.        , -0.76955108, -2.        , -1.69897   ,
       -1.22184875, -0.95860731, -1.39794001, -1.15490196, -0.92081875,
       -0.65757732, -0.69897   , -0.49485002, -1.09691001, -0.60205999,
       -0.85387196, -0.88605665, -0.48148606, -0.43179828, -0.55284197,
       -0.40893539, -0.33724217, -0.35654732, -0.36653154, -0.16115091,
       -0.45593196, -0.39794001, -0.27572413, -0.25181197, -0.24412514,
       -0.20760831, -0.19382003, -0.22184875, -0.28399666, -0.18045606,
       -0.20065945, -0.1079054 , -0.05060999, -0.09691001, -0.10237291,
       -0.03151705, -0.04575749, -0.06550155, -0.02687215, -1.32155166])
```

In [64]:
```
I = [ 'log_JJA','log_SON'],
for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [65]:
```
df.isnull().sum()
```

Out[65]:
```
Year       0
Jan        0
Feb        0
Mar        0
Apr        0
May        0
Jun        0
Jul        0
Aug        0
Sep        0
Oct        0
Nov        0
Dec        0
J-D        0
D-N        0
DJF        0
MAM        0
JJA        0
SON        0
log_SON    0
log_JJA    0
dtype: int64
```

In [66]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.06
Root mean square error:  0.00010701545778834731
R2 square:  0.9557894491843832
Mean absolute Error:  0.0
Root mean square error:  0.0
R2 square:  1.0
```

In [67]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02030907897634135
Root mean square error:  6.24016704780514e-05
R2 square:  0.9817498507366594
Mean absolute Error:  0.001125124606292105
Root mean square error:  2.948966455488963e-37
R2 square:  0.9999829494775933
```

In [68]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.044145165179647936
Root mean square error:  1.7062656791417934e-05
R2 square:  0.9607934585831255
Mean absolute Error:  2.7755575615628915e-18
Root mean square error:  1.4562797311056605e-38
R2 square:  1.0
```

In [69]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.017055172413793084
Root mean square error:  3.2648751486325633e-06
R2 square:  0.9962053955314342
Mean absolute Error:  0.00786323502584367
Root mean square error:  8.138018198439039e-08
R2 square:  0.9988651099621343
```

In [70]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.028796988332976947
Root mean square error:  4.727701698891427e-05
R2 square:  0.9878695216067034
Mean absolute Error:  0.015391644863819987
Root mean square error:  4.477507633235834e-06
R2 square:  0.9971331661829038
```

In [ ]:

In [71]:
```
df['Sqrt_Feature'] = np.sqrt(df['JJA'])
df['Sqrt_Feature2'] = np.sqrt(df['SON'])

df
```

```
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[71]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.1
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.0
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.1
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.2
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	0.9
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.0
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.8
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.9
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 23 columns

In [72]:
```
I = ['Sqrt_Feature', 'Sqrt_Feature2' ],
for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [73]:
```
df['Sqrt_Feature'].unique()
df['Sqrt_Feature2'].unique()
```

Out[73]:
```
array([0.53676175, 0.1       , 0.3       , 0.28284271, 0.37416574,
       0.4472136 , 0.17320508, 0.4       , 0.46904158, 0.38729833,
       0.2       , 0.24494897, 0.        , 0.14142136, 0.31622777,
       0.26457513, 0.50990195, 0.45825757, 0.41231056, 0.52915026,
       0.56568542, 0.53851648, 0.6164414 , 0.58309519, 0.36055513,
       0.6244998 , 0.64807407, 0.76811457, 0.6       , 0.76157731,
       0.79372539, 0.78102497, 0.85440037, 0.83066239, 0.81240384,
       0.84852814, 0.77459667, 0.87749644, 0.88317609, 1.        ,
       0.9539392 , 0.92195445, 0.93808315, 0.98994949, 0.99498744,
       0.9797959 , 0.93273791, 0.26993913])
```

In [74]:
```
df.isnull().sum()
```

Out[74]:
```
Year            0
Jan             0
Feb             0
Mar             0
Apr             0
May             0
Jun             0
Jul             0
Aug             0
Sep             0
Oct             0
Nov             0
Dec             0
J-D             0
D-N             0
DJF             0
MAM             0
JJA             0
SON             0
log_SON         0
log_JJA         0
Sqrt_Feature    0
Sqrt_Feature2   0
dtype: int64
```

In [75]:
```
df.drop('log_SON' ,axis=1,inplace=True)
```

In [76]:
```
df.drop('log_JJA' ,axis=1,inplace=True)
```

In [77]:
```
df
```

Out[77]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [78]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04206896551724139
Root mean square error:  5.7550535077289046e-05
R2 square:  0.9830537772004484
Mean absolute Error:  0.0
Root mean square error:  0.0
R2 square:  1.0
```

In [79]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02025412812282437
Root mean square error:  3.533870558578186e-05
R2 square:  0.9929305561170368
Mean absolute Error:  0.0014230683357758613
Root mean square error:  1.809518583381977e-35
R2 square:  0.9999772953887731
```

In [80]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.03344104171690379
Root mean square error:  0.00038604147820781346
R2 square:  0.9881929893222843
Mean absolute Error:  1.4481169886415085e-18
Root mean square error:  2.0970428127921508e-36
R2 square:  1.0
```

In [81]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.022031613214371727
Root mean square error:  8.214119910505642e-07
R2 square:  0.9950855411233689
Mean absolute Error:  0.007582511401641767
Root mean square error:  8.546549953542641e-07
R2 square:  0.9990145465229348
```

In [82]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.026244077053880207
Root mean square error:  1.1703719004341533e-05
R2 square:  0.9888490703826704
Mean absolute Error:  0.01843426930580561
Root mean square error:  2.435536890895897e-07
R2 square:  0.9959563395395999
```

In [83]:
```
df.drop('Sqrt_Feature' ,axis=1,inplace=True)
df.drop('Sqrt_Feature2' ,axis=1,inplace=True)
```

In [84]:
```
df['Log_Base10_Feature1'] = np.log10(df['JJA'])
df['Log_Base10_Feature2'] = np.log10(df['SON'])
```

```
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: divide by zero encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: divide by zero encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\rayap\anaconda3\lib\site-packages\pandas\core\arraylike.py:402: R
untimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [85]:
```
df
```

Out[85]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [86]:
```
df.isnull().sum()
```

Out[86]:
```
Year                   0
Jan                    0
Feb                    0
Mar                    0
Apr                    0
May                    0
Jun                    0
Jul                    0
Aug                    0
Sep                    0
Oct                    0
Nov                    0
Dec                    0
J-D                    0
D-N                    0
DJF                    0
MAM                    0
JJA                    0
SON                    0
Log_Base10_Feature1    79
Log_Base10_Feature2    76
dtype: int64
```

In [87]:
```
df['Log_Base10_Feature1'].unique()
```

Out[87]:
```
array([        nan, -1.        , -0.76955108, -2.        , -1.69897   ,
                -inf, -1.22184875, -0.95860731, -1.39794001, -1.15490196,
       -0.92081875, -0.65757732, -0.69897   , -0.49485002, -1.09691001,
       -0.60205999, -0.85387196, -0.88605665, -0.48148606, -0.43179828,
       -0.55284197, -0.40893539, -0.33724217, -0.35654732, -0.36653154,
       -0.16115091, -0.45593196, -0.39794001, -0.27572413, -0.25181197,
       -0.24412514, -0.20760831, -0.19382003, -0.22184875, -0.28399666,
       -0.18045606, -0.20065945, -0.1079054 , -0.05060999, -0.09691001,
       -0.10237291, -0.03151705, -0.04575749, -0.06550155, -0.02687215,
       -1.32155166])
```

In [88]:
```
df['Log_Base10_Feature2'].unique()
```

Out[88]:
```
array([        nan, -2.        , -1.04575749, -1.09691001, -0.85387196,
       -0.69897   , -1.52287875, -0.79588002, -0.65757732, -0.82390874,
       -1.39794001, -1.22184875,        -inf, -1.69897   , -1.        ,
       -1.15490196, -0.58502665, -0.67778071, -0.76955108, -0.55284197,
       -0.49485002, -0.537602  , -0.4202164 , -0.46852108, -0.88605665,
       -0.40893539, -0.37675071, -0.22914799, -0.4436975 , -0.23657201,
       -0.20065945, -0.21467016, -0.13667714, -0.16115091, -0.18045606,
       -0.1426675 , -0.22184875, -0.11350927, -0.1079054 ,  0.        ,
       -0.04095861, -0.07058107, -0.05551733, -0.00877392, -0.00436481,
       -0.01772877, -0.06048075, -1.13746832])
```

In [89]:
```
column_name = 'Log_Base10_Feature1'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [90]:
```
column_name = 'Log_Base10_Feature2'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [91]:
```
I = ['Log_Base10_Feature1', 'Log_Base10_Feature2' ],
for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [92]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.06206896551724137
Root mean square error:  0.00045707491082045163
R2 square:  0.9384862765893753
Mean absolute Error:  1.6291316122216971e-18
Root mean square error:  2.654069809940066e-36
R2 square:  1.0
```

In [93]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.015371711738115081
Root mean square error:  5.867162771095373e-06
R2 square:  0.9976908600432179
Mean absolute Error:  0.0013234935742239896
Root mean square error:  1.062243571920944e-34
R2 square:  0.9999784778754719
```

In [94]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04206173137207621
Root mean square error:  0.000251375787180648
R2 square:  0.974473058101505
Mean absolute Error:  1.3274405729213829e-18
Root mean square error:  1.762098474637849e-36
R2 square:  1.0
```

In [95]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024611984567156877
Root mean square error:  4.301097917158105e-06
R2 square:  0.9946979441377531
Mean absolute Error:  0.00879381574946791
Root mean square error:  8.252212613221877e-08
R2 square:  0.99874413642523
```

In [96]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.029794605159445056
Root mean square error:  5.33945912374011e-06
R2 square:  0.9935143816386728
Mean absolute Error:  0.017483636675718887
Root mean square error:  3.260481008550426e-06
R2 square:  0.995135842486089
```

In [97]:
```
df.drop('Log_Base10_Feature1',axis=1,inplace=True)
df.drop('Log_Base10_Feature2',axis=1,inplace=True)
df
```

Out[97]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [98]:
```
df['Squared_Feature1'] = df['SON'] ** 2
df['Squared_Feature2'] = df['JJA'] ** 2
df
```

Out[98]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [ ]:
```
```

In [99]:
```
df.isnull().sum()
```

Out[99]:
```
Year               0
Jan                0
Feb                0
Mar                0
Apr                0
May                0
Jun                0
Jul                0
Aug                0
Sep                0
Oct                0
Nov                0
Dec                0
J-D                0
D-N                0
DJF                0
MAM                0
JJA                0
SON                0
Squared_Feature1   0
Squared_Feature2   0
dtype: int64
```

In [100]:
```
df['Squared_Feature2'].unique()

df['Squared_Feature1'].unique()
```

Out[100]:
```
array([4.00000000e-02, 3.61000000e-02, 8.41000000e-02, 6.25000000e-02,
       7.29000000e-02, 9.00000000e-04, 7.84000000e-02, 1.22500000e-01,
       5.29000000e-02, 5.76000000e-02, 6.76000000e-02, 1.69000000e-02,
       4.00000000e-04, 9.61000000e-02, 1.00000000e-04, 2.20900000e-01,
       1.36900000e-01, 2.89000000e-02, 1.15600000e-01, 1.93600000e-01,
       2.02500000e-01, 9.00000000e-02, 2.70400000e-01, 1.21000000e-02,
       1.52100000e-01, 1.44000000e-02, 2.56000000e-02, 1.00000000e-02,
       3.60000000e-03, 2.25000000e-02, 8.10000000e-03, 6.40000000e-03,
       1.96000000e-02, 4.84000000e-02, 4.90000000e-03, 1.60000000e-03,
       0.00000000e+00, 2.50000000e-03, 4.41000000e-02, 1.02400000e-01,
       1.44400000e-01, 1.76400000e-01, 3.48100000e-01, 1.29600000e-01,
       3.36400000e-01, 3.96900000e-01, 3.72100000e-01, 5.32900000e-01,
       4.76100000e-01, 4.35600000e-01, 5.18400000e-01, 3.60000000e-01,
       5.92900000e-01, 6.08400000e-01, 1.00000000e+00, 8.28100000e-01,
       7.22500000e-01, 7.74400000e-01, 9.60400000e-01, 9.80100000e-01,
       9.21600000e-01, 7.56900000e-01, 5.30961905e-03])
```

In [101]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.03310344827586208
Root mean square error:  8.038049940546961e-05
R2 square:  0.9851276015628747
Mean absolute Error:  6.033820786006285e-19
Root mean square error:  3.6406993277641506e-37
R2 square:  1.0
```

In [102]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024647259321847522
Root mean square error:  8.539387705469293e-07
R2 square:  0.9915967417238086
Mean absolute Error:  0.001069493788148569
Root mean square error:  2.390631082020132e-36
R2 square:  0.9999859085144742
```

In [103]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02758620689655173
Root mean square error:  7.609988109393535e-06
R2 square:  0.9877087799719364
Mean absolute Error:  0.0
Root mean square error:  0.0
R2 square:  1.0
```

In [104]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02866428743670118
Root mean square error:  0.0003471720556409876
R2 square:  0.9776081525522313
Mean absolute Error:  0.0077815141380358265
Root mean square error:  1.8010204371228414e-07
R2 square:  0.9991711808309415
```

In [105]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.032267131591373135
Root mean square error:  1.8500373046244434e-06
R2 square:  0.9903016744942968
Mean absolute Error:  0.018597595613642454
Root mean square error:  1.624435319756588e-06
R2 square:  0.995593614898251
```

In [106]:
```
df.drop('Squared_Feature2',axis=1,inplace=True)
df.drop('Squared_Feature1',axis=1,inplace=True)
```

In [107]:
```
df
```

Out[107]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [108]:
```
df['Squared_Feature1'] = df['SON'] ** 3
df['Squared_Feature2'] = df['JJA'] ** 3

df
```

Out[108]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [109]:
```
df.isnull().sum()
```

Out[109]:
```
Year               0
Jan                0
Feb                0
Mar                0
Apr                0
May                0
Jun                0
Jul                0
Aug                0
Sep                0
Oct                0
Nov                0
Dec                0
J-D                0
D-N                0
DJF                0
MAM                0
JJA                0
SON                0
Squared_Feature1   0
Squared_Feature2   0
dtype: int64
```

In [110]:
```
df['Squared_Feature2'].unique()

df['Squared_Feature1'].unique()
```

Out[110]:
```
array([-8.00000000e-03, -6.85900000e-03, -2.43890000e-02, -1.56250000e-02,
       -1.96830000e-02, -2.70000000e-05, -2.19520000e-02, -4.28750000e-02,
       -1.21670000e-02, -1.38240000e-02, -1.75760000e-02, -2.19700000e-03,
       -8.00000000e-06, -2.97910000e-02,  1.00000000e-06, -1.00000000e-06,
       -1.03823000e-01, -5.06530000e-02, -4.91300000e-03, -3.93040000e-02,
       -8.51840000e-02, -9.11250000e-02, -2.70000000e-02, -1.40608000e-01,
       -1.33100000e-03, -5.93190000e-02, -1.72800000e-03, -4.09600000e-03,
       -1.00000000e-03, -2.16000000e-04, -3.37500000e-03,  7.29000000e-04,
        5.12000000e-04,  2.74400000e-03,  8.00000000e-03,  2.70000000e-05,
        4.09600000e-03,  1.06480000e-02,  3.37500000e-03, -3.43000000e-04,
       -1.06480000e-02,  6.40000000e-05, -2.74400000e-03,  2.16000000e-04,
        0.00000000e+00,  8.00000000e-06, -7.29000000e-04, -1.25000000e-04,
        1.00000000e-03,  3.43000000e-04,  1.75760000e-02,  9.26100000e-03,
        4.91300000e-03,  2.19520000e-02,  3.27680000e-02,  2.43890000e-02,
        5.48720000e-02,  3.93040000e-02,  2.19700000e-03,  5.93190000e-02,
        7.40880000e-02,  2.05379000e-01,  4.66560000e-02,  1.95112000e-01,
        2.50047000e-01,  2.26981000e-01,  3.89017000e-01,  3.28509000e-01,
        2.87496000e-01,  3.73248000e-01,  2.16000000e-01,  4.56533000e-01,
        4.74552000e-01,  1.00000000e+00,  7.53571000e-01,  6.14125000e-01,
        6.81472000e-01,  9.41192000e-01,  9.70299000e-01,  8.84736000e-01,
        6.58503000e-01,  3.86896717e-04])
```

In [111]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.029662406558958278
Root mean square error:  3.843581395530035e-05
R2 square:  0.9848613508111667
Mean absolute Error:  2.3531901065424513e-18
Root mean square error:  8.19157348746934e-37
R2 square:  1.0
```

In [112]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.020003956258872406
Root mean square error:  0.00010383709132849356
R2 square:  0.9928967989167479
Mean absolute Error:  0.001131019497277428
Root mean square error:  2.245952535031134e-34
R2 square:  0.9999839362979949
```

In [113]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02793103448275862
Root mean square error:  1.4387633769322206e-05
R2 square:  0.975709054075113
Mean absolute Error:  1.3274405729213829e-18
Root mean square error:  1.762098474637849e-36
R2 square:  1.0
```

In [114]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.020392163009404374
Root mean square error:  1.0951199800507116e-05
R2 square:  0.9951667731587543
Mean absolute Error:  0.007947856491334757
Root mean square error:  3.6278554397071417e-07
R2 square:  0.998950323580824
```

In [115]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.027988011019312534
Root mean square error:  6.57837976648749e-07
R2 square:  0.9863663414765149
Mean absolute Error:  0.01665626339633603
Root mean square error:  2.1538198078685803e-07
R2 square:  0.9966152826153541
```

In [116]:
```
df.drop('Squared_Feature2',axis=1,inplace=True)
df.drop('Squared_Feature1',axis=1,inplace=True)
df
```

Out[116]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [118]:
```
df['Exponential_Feature1'] = np.exp(df['JJA'])
df['Exponential_Feature2'] = np.exp(df['SON'])
df
```

Out[118]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [119]:
```
df.isnull().sum()
```

Out[119]:
```
Year                  0
Jan                   0
Feb                   0
Mar                   0
Apr                   0
May                   0
Jun                   0
Jul                   0
Aug                   0
Sep                   0
Oct                   0
Nov                   0
Dec                   0
J-D                   0
D-N                   0
DJF                   0
MAM                   0
JJA                   0
SON                   0
Exponential_Feature1  0
Exponential_Feature2  0
dtype: int64
```

In [120]:
```
df['Exponential_Feature1'].unique()
```

Out[120]:
```
array([0.84366482, 0.92311635, 0.85214379, 0.90483742, 0.72614904,
       0.69767633, 0.75578374, 0.74826357, 0.86935824, 0.87809543,
       0.74081822, 0.83527021, 0.76337949, 0.81058425, 0.82695913,
       0.94176453, 0.7945336 , 0.86070798, 0.66365025, 0.60653066,
       0.77880078, 0.69073433, 0.67705687, 0.65050909, 0.63762815,
       0.68386141, 0.71892373, 0.73344696, 0.77105159, 0.8025188 ,
       0.95122942, 0.88692044, 0.98019867, 1.10517092, 1.18530485,
       1.01005017, 1.02020134, 0.96078944, 0.91393119, 1.        ,
       1.06183655, 1.11627807, 0.97044553, 1.04081077, 1.07250818,
       1.12749685, 0.89583414, 1.24607673, 1.22140276, 1.37712776,
       1.08328707, 1.28402542, 1.1502738 , 1.13882838, 1.39096813,
       1.44773461, 1.32312981, 1.47698079, 1.58407398, 1.55270722,
       1.53725752, 1.99371553, 1.41906755, 1.4918247 , 1.69893231,
       1.7506725 , 1.76826705, 1.85892804, 1.89648088, 1.8221188 ,
       1.68202765, 1.93479233, 1.87761058, 2.18147227, 2.43512965,
       2.22554093, 2.20339643, 2.53450918, 2.45960311, 2.36316069,
       2.55998142, 1.04884788])
```

In [121]:
```
df['Exponential_Feature2'].unique()
```

Out[121]:
```
array([0.81873075, 0.82695913, 0.74826357, 0.77880078, 0.76337949,
       0.97044553, 0.75578374, 0.70468809, 0.7945336 , 0.78662786,
       0.77105159, 0.87809543, 0.98019867, 0.73344696, 1.01005017,
       0.99004983, 0.62500227, 0.69073433, 0.84366482, 0.71177032,
       0.64403642, 0.63762815, 0.74081822, 0.59452055, 0.89583414,
       0.67705687, 0.88692044, 0.85214379, 0.90483742, 0.94176453,
       0.86070798, 1.09417428, 1.08328707, 1.1502738 , 1.22140276,
       1.03045453, 1.17351087, 1.24607673, 1.16183424, 0.93239382,
       0.8025188 , 1.04081077, 0.86935824, 1.06183655, 1.        ,
       1.02020134, 0.91393119, 0.95122942, 1.10517092, 1.07250818,
       1.29693009, 1.23367806, 1.18530485, 1.32312981, 1.37712776,
       1.33642749, 1.46228459, 1.40494759, 1.13882838, 1.47698079,
       1.52196156, 1.80398842, 1.43332941, 1.78603843, 1.87761058,
       1.8404314 , 2.07508061, 1.99371553, 1.93479233, 2.05443321,
       1.8221188 , 2.15976625, 2.18147227, 2.71828183, 2.48432253,
       2.33964685, 2.41089971, 2.66445624, 2.69123447, 2.61169647,
       2.38691085, 1.07558762])
```

In [122]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04413793103448277
Root mean square error:  6.848989298454222e-05
R2 square:  0.9708238431641838
Mean absolute Error:  1.4481169886415085e-18
Root mean square error:  2.0970428127921508e-36
R2 square:  1.0
```

In [123]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.01951971225832264
Root mean square error:  6.850301772800749e-05
R2 square:  0.9966403717313005
Mean absolute Error:  0.0010555611678160683
Root mean square error:  1.0645768904315156e-34
R2 square:  0.9999854158339825
```

In [124]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024834820352061727
Root mean square error:  3.0360179720332014e-05
R2 square:  0.9916570085035118
Mean absolute Error:  2.2325136908223255e-18
Root mean square error:  1.0521621057238397e-36
R2 square:  1.0
```

In [125]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02125143477212444
Root mean square error:  6.403328346980955e-06
R2 square:  0.9931757366523107
Mean absolute Error:  0.008083636363636357
Root mean square error:  1.880325520199938e-06
R2 square:  0.9989390020857571
```

In [126]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.023015570109287318
Root mean square error:  1.8792313762891227e-06
R2 square:  0.991888024498217
Mean absolute Error:  0.0188808954765816
Root mean square error:  9.420027161275422e-06
R2 square:  0.9958538774044163
```

In [127]:
```
df.drop('Exponential_Feature1',axis=1,inplace=True)
df.drop('Exponential_Feature2',axis=1,inplace=True)
```

In [128]:
```
df
```

Out[128]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	C
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	-0.2400
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	-0.2200
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	-0.2400
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	-0.1200
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	-0.2500
...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	1.0100
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	0.8900
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	1.0000
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	0.9700
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	0.0841

144 rows × 19 columns

In [129]:
```
df['Reciprocal_Feature'] = 1 / df['SON']
df['Reciprocal_Feature2'] = 1 / df['JJA']
df
```

Out[129]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	...	
0	1880	-0.19	-0.25	-0.09	-0.17	-0.10	-0.210000	-0.180000	-0.110000	-0.150000	...	-0.2
1	1881	-0.20	-0.15	0.03	0.05	0.05	-0.190000	0.000000	-0.040000	-0.160000	...	-0.1
2	1882	0.16	0.13	0.04	-0.16	-0.14	-0.220000	-0.170000	-0.080000	-0.150000	...	-0.1
3	1883	-0.30	-0.37	-0.13	-0.19	-0.18	-0.080000	-0.080000	-0.140000	-0.230000	...	-0.2
4	1884	-0.13	-0.09	-0.37	-0.40	-0.34	-0.350000	-0.310000	-0.280000	-0.280000	...	-0.3
...	...	...	...	...	...	...	...	...	...	...	...	
139	2019	0.93	0.95	1.17	1.02	0.85	0.910000	0.940000	0.950000	0.930000	...	1.0
140	2020	1.18	1.25	1.17	1.13	1.02	0.920000	0.900000	0.880000	0.990000	...	1.1
141	2021	0.81	0.64	0.89	0.76	0.79	0.840000	0.920000	0.820000	0.930000	...	0.9
142	2022	0.91	0.90	1.05	0.84	0.84	0.930000	0.940000	0.960000	0.900000	...	0.7
143	2023	0.87	0.98	1.21	1.00	0.94	0.033147	0.055874	0.054406	0.058182	...	0.0

144 rows × 21 columns

In [130]:
```
df['Reciprocal_Feature'].unique()
df['Reciprocal_Feature2'].unique()
```

Out[130]:
```
array([ -5.88235294, -12.5       ,  -6.25      , -10.        ,
         -3.125     ,  -2.77777778,  -3.57142857,  -3.44827586,
         -7.14285714,  -7.69230769,  -3.33333333,  -5.55555556,
         -3.7037037 ,  -4.76190476,  -5.26315789, -16.66666667,
         -4.34782609,  -6.66666667,  -2.43902439,  -2.        ,
         -4.        ,  -2.7027027 ,  -2.56410256,  -2.3255814 ,
         -2.22222222,  -2.63157895,  -3.03030303,  -3.22580645,
         -3.84615385,  -4.54545455, -20.        ,  -8.33333333,
        -50.        ,  10.        ,   5.88235294, 100.        ,
         50.        , -25.        , -11.11111111,          inf,
         16.66666667,   9.09090909, -33.33333333,  25.        ,
         14.28571429,   8.33333333,  -9.09090909,   4.54545455,
          5.        ,   3.125     ,  12.5       ,   4.        ,
          7.14285714,   7.69230769,   3.03030303,   2.7027027 ,
          3.57142857,   2.56410256,   2.17391304,   2.27272727,
          2.3255814 ,   1.44927536,   2.85714286,   2.5       ,
          1.88679245,   1.78571429,   1.75438596,   1.61290323,
          1.5625    ,   1.66666667,   1.92307692,   1.51515152,
          1.58730159,   1.28205128,   1.12359551,   1.25      ,
          1.26582278,   1.07526882,   1.11111111,   1.1627907 ,
          1.06382979,  20.96774194])
```

In [131]:
```
column_name = 'Reciprocal_Feature2'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [132]:
```
column_name = 'Reciprocal_Feature'
df[column_name] = df[column_name].replace([np.inf, -np.inf], np.nan)
```

In [133]:
```
df.isnull().sum()
```

Out[133]:
```
Year                 0
Jan                  0
Feb                  0
Mar                  0
Apr                  0
May                  0
Jun                  0
Jul                  0
Aug                  0
Sep                  0
Oct                  0
Nov                  0
Dec                  0
J-D                  0
D-N                  0
DJF                  0
MAM                  0
JJA                  0
SON                  0
Reciprocal_Feature   1
Reciprocal_Feature2  2
dtype: int64
```

In [134]:
```
I = ['Reciprocal_Feature', 'Reciprocal_Feature2' ],
for i in I:
    df[i] = df[i].fillna(df[i].mean())
```

In [135]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.03310344827586207
Root mean square error:  7.609988109393585e-06
R2 square:  0.986790200275688
Mean absolute Error:  1.6291316122216971e-18
Root mean square error:  2.654069809940066e-36
R2 square:  1.0
```

In [136]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import GradientBoostingRegressor
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)
regressor = GradientBoostingRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =GradientBoostingRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.024601316799558175
Root mean square error:  0.00012493972833923389
R2 square:  0.9934372120510324
Mean absolute Error:  0.001126265576967551
Root mean square error:  5.466737584345858e-36
R2 square:  0.999984106350767
```

In [137]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.tree import DecisionTreeRegressor
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)
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =DecisionTreeRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.04586206896551724
Root mean square error:  0.00019988109393579066
R2 square:  0.9610654006073911
Mean absolute Error:  9.050731179009428e-19
Root mean square error:  2.948966455488963e-37
R2 square:  1.0
```

In [138]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import RandomForestRegressor
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)
regressor = RandomForestRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 = RandomForestRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.02397248613455513
Root mean square error:  4.392777567218731e-05
R2 square:  0.9855433773986004
Mean absolute Error:  0.007432076619033115
Root mean square error:  2.225923136733689e-07
R2 square:  0.9993659207503436
```

In [139]:
```
x=df.drop('J-D',axis=1)
y=df['J-D']
from sklearn.ensemble import AdaBoostRegressor
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)
regressor = AdaBoostRegressor()
regressor.fit(x_train, y_train)
y_pred =regressor.predict(x_test)
y_pred
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred-y_test)))
print("Root mean square error: " ,np.mean(y_pred-y_test)**2)
print("R2 square: " ,r2_score(y_pred,y_test))
regressor2 =AdaBoostRegressor()
regressor2.fit(x_train, y_train)
y_pred1 =regressor2.predict(x_train)
y_pred1
from sklearn.metrics import r2_score
print("Mean absolute Error: " ,np.mean(np.absolute(y_pred1-y_train)))
print("Root mean square error: " ,np.mean(y_pred1-y_train)**2)
print("R2 square: " ,r2_score(y_pred1,y_train))
```

```
Mean absolute Error:  0.028212206425020328
Root mean square error:  3.174111552757043e-05
R2 square:  0.9911941619200128
Mean absolute Error:  0.017969914948030987
Root mean square error:  1.878235159692435e-09
R2 square:  0.9957464109010788
```

In [ ]: