In [155]:

```
import pandas as pd
import numpy as np
cc=pd.read_csv('covid_country.csv')
```

In [156]:

```
cc
```

Out[156]:

	Date	Country	Confirmed	Recovered	Deaths
0	2020-01-22	Afghanistan	0	0	0
1	2020-01-22	Albania	0	0	0
2	2020-01-22	Algeria	0	0	0
3	2020-01-22	Andorra	0	0	0
4	2020-01-22	Angola	0	0	0
...	...	...	...	...	...
23683	2020-05-26	West Bank and Gaza	429	365	3
23684	2020-05-26	Western Sahara	9	6	1
23685	2020-05-26	Yemen	249	10	49
23686	2020-05-26	Zambia	920	336	7
23687	2020-05-26	Zimbabwe	56	25	4

23688 rows × 5 columns

In [157]:

```
cc.dtypes
```

Out[157]:

```
Date          object
Country       object
Confirmed      int64
Recovered      int64
Deaths         int64
dtype: object
```

In [158]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cc.isnull())
plt.show()
```

In [159]:

```
cc.columns
```

Out[159]:

```
Index(['Date', 'Country', 'Confirmed', 'Recovered', 'Deaths'], dtype='object')
```

In [160]:

```
cc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23688 entries, 0 to 23687
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Date       23688 non-null  object
 1   Country    23688 non-null  object
 2   Confirmed  23688 non-null  int64 
 3   Recovered  23688 non-null  int64 
 4   Deaths     23688 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 925.4+ KB
```

In [161]:

```
cc.describe()
```

Out[161]:

	Confirmed	Recovered	Deaths
count	2.368800e+04	23688.000000	23688.000000
mean	7.969368e+03	2581.801714	526.935030
std	5.842109e+04	15143.101257	3992.815956
min	0.000000e+00	0.000000	0.000000
25%	0.000000e+00	0.000000	0.000000
50%	1.800000e+01	1.000000	0.000000
75%	7.300000e+02	123.000000	13.000000
max	1.680913e+06	384902.000000	98913.000000

In [162]:

```
cc_df=cc[["year","month","day"]]=cc["Date"].str.split("-",expand=True)
```

In [163]:

```
cc_df
```

Out[163]:

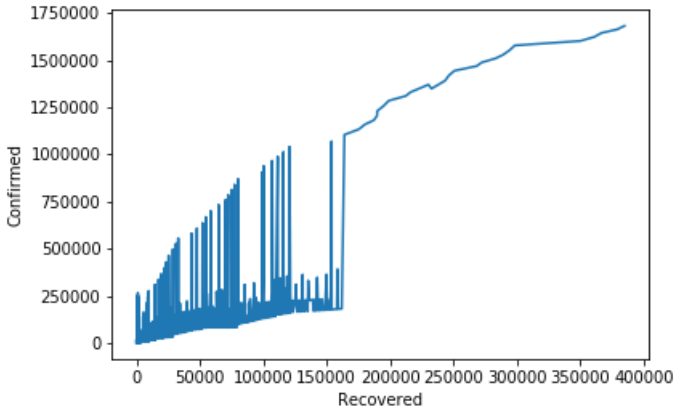
	0	1	2
0	2020	01	22
1	2020	01	22
2	2020	01	22
3	2020	01	22
4	2020	01	22
...	...	...	...
23683	2020	05	26

	0	1	2
23684	2020	05	26
23685	2020	05	26
23686	2020	05	26
23687	2020	05	26

23688 rows × 3 columns

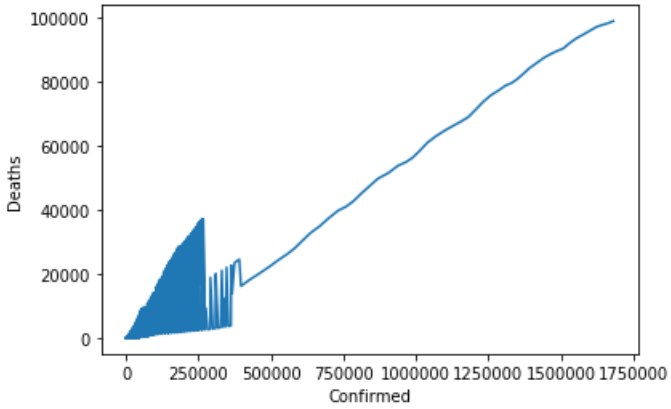
In [164]:

```
sns.lineplot(x="Recovered",y="Confirmed",data=cc)
plt.show()
```



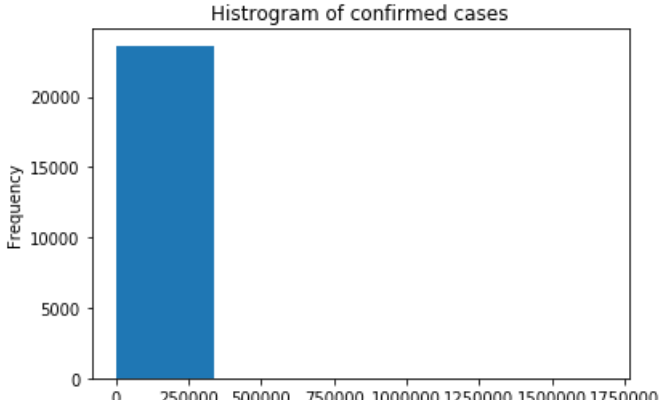
In [165]:

```
sns.lineplot(x="Confirmed",y="Deaths",data=cc)
plt.show()
```



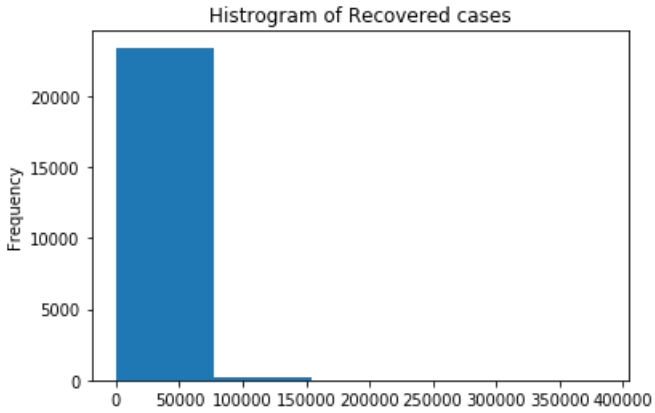
In [166]:

```
cc['Confirmed'].plot.hist(bins=5)
plt.title("Histrogram of confirmed cases")
plt.show()
```



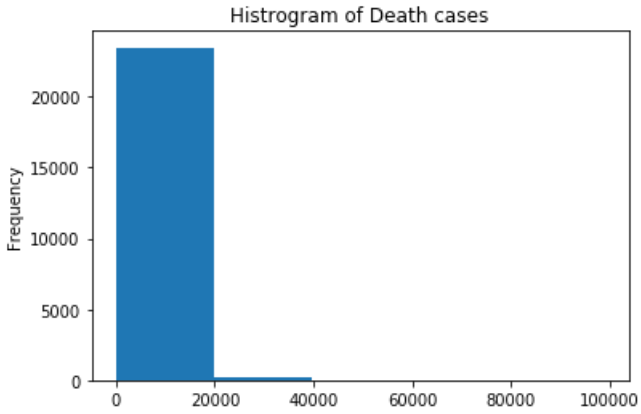
In [167]:

```
cc['Recovered'].plot.hist(bins=5)
plt.title("Histrogram of Recovered cases")
plt.show()
```



In [199]:

```
cc['Deaths'].plot.hist(bins=5)
plt.title("Histrogram of Death cases")
plt.show()
```

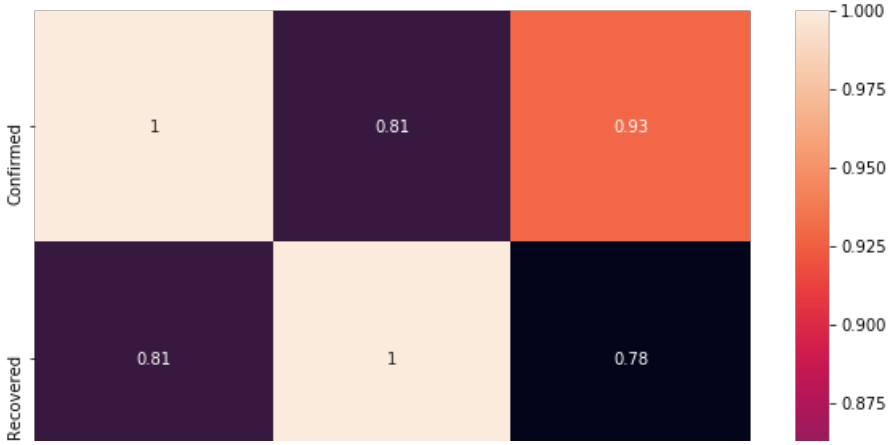


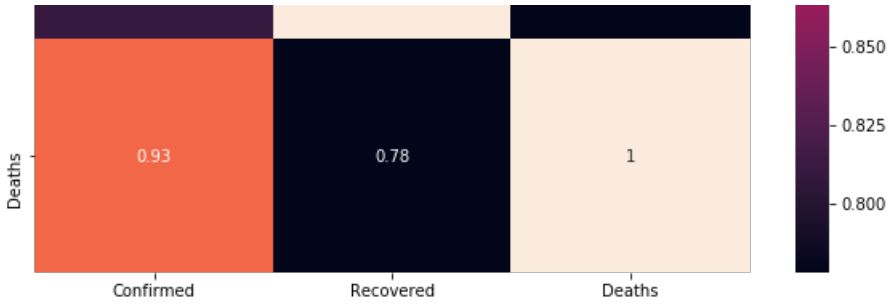
In [169]:

```
plt.figure(figsize=(10,8))
sns.heatmap(cc.corr(),annot=True)
```

Out[169]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26edc493388>
```





In [170]:

```
cc.corr()
```

Out[170]:

	Confirmed	Recovered	Deaths
Confirmed	1.000000	0.810991	0.929718
Recovered	0.810991	1.000000	0.778094
Deaths	0.929718	0.778094	1.000000

In [171]:

```
cc.dtypes
```

Out[171]:

```
Date          object
Country       object
Confirmed      int64
Recovered      int64
Deaths         int64
year          object
month         object
day           object
dtype: object
```

In [172]:

```
cc["year"].unique()
```

Out[172]:

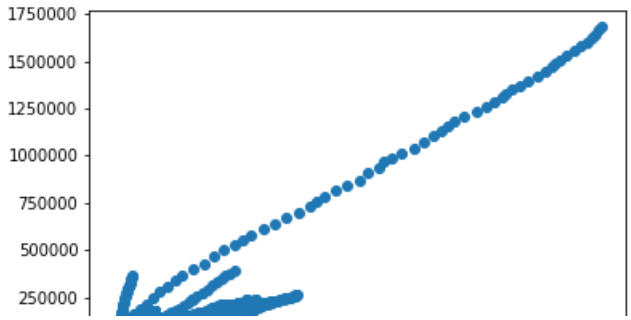
```
array(['2020'], dtype=object)
```

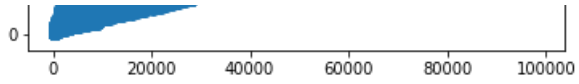
In [174]:

```
cc1=cc.drop(["Date","Country","day","month","year"],axis=1)
```

In [175]:

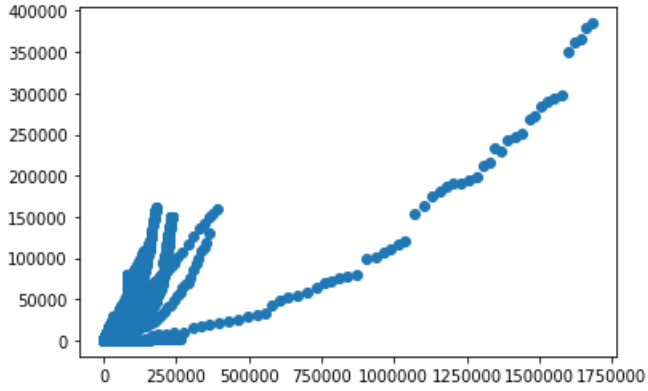
```
plt.scatter(cc['Deaths'],cc['Confirmed'])
plt.show()
```





In [176]:

```
plt.scatter(cc['Confirmed'],cc['Recovered'])
plt.show()
```



In [177]:

```
cc1.shape
```

Out[177]:

```
(23688, 3)
```

In [178]:

```
cc1.head()
```

Out[178]:

	Confirmed	Recovered	Deaths
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

In [179]:

```
cc1.skew()
```

Out[179]:

```
Confirmed    18.674502
Recovered    10.662286
Deaths       14.218167
dtype: float64
```

In [180]:

```
for col in cc1.columns:
    if cc1[col].skew()>0.55:
        cc1[col]=np.log1p(cc1[col])
```

In [181]:

```
cc1.skew()
```

Out[181]:

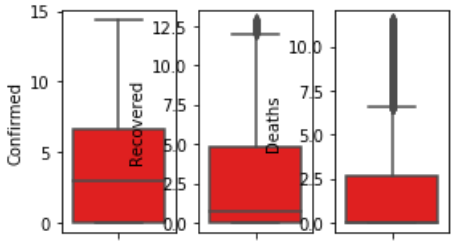
```
Confirmed    0.538470
Recovered    1.011590
Deaths       1.598833
dtype: float64
```

In [182]:

```
collist=cc1.columns.values
ncol=10
nrow=10
```

In [183]:

```
plt.figure(figsize=(15,30))
for i in range(0,len(collist)):
    plt.subplot(nrow,ncol,i+1)
    sns.boxplot(cc1[collist[i]],color='red',orient='v')
```



In [184]:

```
from scipy.stats import zscore
z=np.abs(zscore(cc1))
z
```

Out[184]:

```
array([[1.02851613, 0.8092577 , 0.67086286],
       [1.02851613, 0.8092577 , 0.67086286],
       [1.02851613, 0.8092577 , 0.67086286],
       ...,
       [0.50534411, 0.05256909, 0.96586491],
       [0.86759469, 1.02734898, 0.19914217],
       [0.09464234, 0.21887918, 0.00250018]])
```

In [185]:

```
threshold=3
print(np.where(z>3))
```

```
(array([11741, 11929, 12117, 12305, 12493, 12681, 12754, 12869, 12942,
       13057, 13130, 13245, 13318, 13334, 13433, 13506, 13522, 13598,
       13621, 13694, 13710, 13786, 13809, 13882, 13898, 13974, 13997,
       14070, 14086, 14162, 14185, 14258, 14274, 14350, 14373, 14446,
       14462, 14466, 14538, 14561, 14634, 14650, 14654, 14726, 14749,
       14822, 14838, 14842, 14914, 14937, 15010, 15026, 15030, 15102,
       15125, 15198, 15214, 15218, 15290, 15313, 15386, 15402, 15406,
       15478, 15501, 15574, 15590, 15594, 15666, 15689, 15762, 15778,
       15782, 15854, 15877, 15950, 15966, 15970, 16042, 16065, 16138,
       16154, 16158, 16230, 16253, 16326, 16342, 16346, 16418, 16441,
       16514, 16530, 16534, 16606, 16629, 16702, 16718, 16722, 16794,
       16817, 16890, 16906, 16910, 16982, 17005, 17078, 17094, 17098,
       17170, 17193, 17266, 17282, 17286, 17312, 17358, 17381, 17454,
       17470, 17474, 17500, 17546, 17569, 17642, 17658, 17662, 17688,
       17734, 17757, 17830, 17846, 17850, 17876, 17922, 17945, 18018,
       18034, 18038, 18064, 18110, 18133, 18206, 18222, 18226, 18252,
       18298, 18321, 18394, 18410, 18414, 18440, 18486, 18490, 18509,
       18582, 18598, 18602, 18628, 18674, 18678, 18697, 18770, 18786,
```

```
        18790, 18816, 18862, 18866, 18885, 18958, 18974, 18978, 19004,
        19011, 19050, 19054, 19073, 19146, 19162, 19162, 19166, 19192,
        19199, 19238, 19242, 19261, 19334, 19350, 19350, 19354, 19380,
        19387, 19426, 19430, 19449, 19522, 19538, 19538, 19542, 19568,
        19575, 19614, 19618, 19637, 19710, 19726, 19726, 19730, 19756,
        19763, 19802, 19806, 19825, 19898, 19914, 19914, 19918, 19944,
        19951, 19990, 19994, 20009, 20013, 20086, 20102, 20102, 20106,
        20132, 20139, 20178, 20182, 20197, 20201, 20274, 20290, 20290,
        20294, 20320, 20327, 20366, 20370, 20385, 20389, 20462, 20478,
        20478, 20482, 20508, 20515, 20554, 20558, 20573, 20577, 20650,
        20666, 20666, 20670, 20696, 20703, 20742, 20746, 20761, 20765,
        20838, 20854, 20854, 20858, 20884, 20891, 20930, 20934, 20949,
        20953, 21026, 21042, 21042, 21046, 21072, 21079, 21118, 21122,
        21137, 21141, 21214, 21230, 21230, 21234, 21260, 21267, 21306,
        21310, 21325, 21329, 21402, 21418, 21418, 21422, 21448, 21455,
        21494, 21498, 21513, 21517, 21590, 21606, 21606, 21610, 21636,
        21643, 21682, 21686, 21701, 21705, 21778, 21794, 21794, 21798,
        21824, 21831, 21870, 21874, 21889, 21893, 21966, 21982, 21982,
        21986, 22012, 22019, 22058, 22062, 22077, 22081, 22154, 22170,
        22170, 22174, 22200, 22207, 22246, 22250, 22265, 22269, 22342,
        22358, 22358, 22362, 22388, 22395, 22434, 22438, 22453, 22457,
        22530, 22546, 22546, 22550, 22576, 22583, 22622, 22626, 22641,
        22645, 22673, 22718, 22734, 22734, 22738, 22764, 22771, 22810,
        22814, 22829, 22833, 22861, 22906, 22922, 22922, 22926, 22952,
        22959, 22968, 22998, 23002, 23017, 23021, 23049, 23094, 23110,
        23110, 23114, 23140, 23147, 23156, 23186, 23190, 23205, 23209,
        23237, 23282, 23298, 23298, 23302, 23328, 23335, 23344, 23374,
        23378, 23393, 23397, 23425, 23470, 23486, 23486, 23490, 23516,
        23523, 23532, 23562, 23566, 23581, 23585, 23613, 23658, 23674,
        23674, 23678], dtype=int64), array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
        2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 1, 2, 2], dtype=int64))
```

In [186]:

```
cc_new=cc1[(z<3).all(axis=1)]
```

In [187]:

```
print(cc1.shape,'\t',cc_new.shape)
```

```
(23688, 3)    (23288, 3)
```

In [188]:

```
covid=cc_new
print(covid.shape)
```

```
(23288, 3)
```

In [189]:

```
covid_x=covid.drop(columns=['Deaths'])
```

In [190]:

```
y=pd.DataFrame(covid['Deaths'])
```

In [191]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(covid_x)
x=pd.DataFrame(x,columns=covid_x.columns)
```

In [192]:

```
x.skew()
```

Out[192]:

```
Confirmed    0.489203
Recovered    0.984873
dtype: float64
```

In [193]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
max_r_score=0
for r_state in range (40,3000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=.20)
    regr=linear_model.LinearRegression()
    regr.fit(x_train,y_train)
    y_pred=regr.predict(x_test)
    r_scr=r2_score(y_test,y_pred)
    if r_scr>max_r_score:
        max_r_score=r_scr
        final_r_state=r_state
print("max r2 score corresponding to ",final_r_state,"is",max_r_score)
```

```
max r2 score corresponding to  437 is 0.836365023382512
```

In [ ]:

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
import warnings
warnings.filterwarnings('ignore')
```

In [194]:

```
model=[LinearRegression(),DecisionTreeRegressor(),KNeighborsRegressor(),SVR(),Lasso(),Ridge(),Elast
icNet()]
for m in model:
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=437,test_size=.20)
    m.fit(x_train,y_train)
    print('Score of',m,'is:',m.score(x_train,y_train))
    predm=m.predict(x_test)
    print('Error:')
    print('Mean Absolute Error :',mean_absolute_error(y_test,predm))
    print('Mean Squared Error :',mean_squared_error(y_test,predm))
    print('r2_score',r2_score(y_test,predm))

print('**********************************************************************************
*******')
```

```
    print('\n')
```

```
Score of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False) is: 0.817
1394843971453
Error:
Mean Absolute Error : 0.6549911124945789
Mean Squared Error : 0.77800279301519
r2_score 0.836365023382512
**********************************************************************************************



Score of DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best') is: 0.9945509858026207
Error:
Mean Absolute Error : 0.3621643430077506
Mean Squared Error : 0.5710961937909168
r2_score 0.8798830632019485
**********************************************************************************************



Score of KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform') is: 0.9461092310490143
Error:
Mean Absolute Error : 0.3262952212048879
Mean Squared Error : 0.37550093886759617
r2_score 0.9210220221532749
**********************************************************************************************



Score of SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False) is: 0.9071441870634686
Error:
Mean Absolute Error : 0.40331230546572466
Mean Squared Error : 0.4045603770518489
r2_score 0.9149100383268818
**********************************************************************************************



Score of Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False) is: 0.5885688863667593
Error:
Mean Absolute Error : 1.0423129011935042
Mean Squared Error : 1.9148521989564795
r2_score 0.5972549229703453
**********************************************************************************************



Score of Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001) is: 0.8171394837494403
Error:
Mean Absolute Error : 0.6549802888531777
Mean Squared Error : 0.7780054854083298
r2_score 0.8363644570995983
**********************************************************************************************



Score of ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False) is: 0.6748736109157
756
Error:
Mean Absolute Error : 0.8897115078579853
Mean Squared Error : 1.493054749259247
r2_score 0.6859703060488932
```

****

In [195]:

```
from sklearn.model_selection import cross_val_score
for m in model:
    score=cross_val_score(m,x,y,cv=5,scoring='r2')
    print('Score of',m,'is:',score)
    print('Mean score:',score.mean())
    print('Standard deviation:',score.std())

print('**********
*******')
    print('\n')
```

```
Score of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False) is: [-0.8
3104219  0.41245473  0.70626768  0.72997895  0.70191693]
Mean score: 0.34391521930082547
Standard deviation: 0.5989544743015163
**********



Score of DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best') is: [0.92454186 0.78030606 0.68773853 0.7
1706607 0.71462635]
Mean score: 0.7648557744647092
Standard deviation: 0.08543099795220542
**********



Score of KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform') is: [0.93382393 0.84901597 0.77921399 0.80473999 0.79343412]
Mean score: 0.8320455985727392
Standard deviation: 0.05599007626873273
**********



Score of SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False) is: [0.91979415
0.88082303 0.82493627 0.85471091 0.83455628]
Mean score: 0.8629641272777471
Standard deviation: 0.03426318955667884
**********



Score of Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False) is: [-2.28466422 -0.27070963  0.52152564
0.28357683 -0.22430264]
Mean score: -0.39491480456366684
Standard deviation: 0.9915545669483996
**********



Score of Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001) is: [-0.83050286  0.41254673  0
.70627985  0.72997805  0.70192572]
Mean score: 0.3440454971948693
Standard deviation: 0.5987473890250047
**********
```

```
Score of ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False) is: [-0.49082117  0
.16871162  0.58654492  0.44668477  0.21240342]
Mean score: 0.18470471206298117
Standard deviation: 0.3708011086586259
**********************************************************************************************
```

In [196]:

```
#saving the KNeighborsRegressor
import joblib
joblib.dump(KNeighborsRegressor,'covid.pkl')
```

Out[196]:

```
['covid.pkl']
```

In [ ]: