Happiness_Score_Prediction:-

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import sklearn

import warnings
warnings.filterwarnings('ignore')
```

Importing the Dataset:-

In [18]:

```
hp=pd.read_csv('Happiness_Dataset.csv')
hp
```

Out[18]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	F
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	
...	...	...	...	...	...	...	...	...	
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	

158 rows × 12 columns

In [19]:

```
hp.head()
```

Out[19]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Free
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.6
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.6
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.6
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.6
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.6

In [20]:

```
hp.shape
```

Out[20]:

```
(158, 12)
```

In [21]:

```
hp.columns
```

Out[21]:

```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [22]:

```
hp.Country.unique()
```

Out[22]:

```
array(['Switzerland', 'Iceland', 'Denmark', 'Norway', 'Canada', 'Finland',
       'Netherlands', 'Sweden', 'New Zealand', 'Australia', 'Israel',
       'Costa Rica', 'Austria', 'Mexico', 'United States', 'Brazil',
       'Luxembourg', 'Ireland', 'Belgium', 'United Arab Emirates',
       'United Kingdom', 'Oman', 'Venezuela', 'Singapore', 'Panama',
       'Germany', 'Chile', 'Qatar', 'France', 'Argentina',
       'Czech Republic', 'Uruguay', 'Colombia', 'Thailand',
       'Saudi Arabia', 'Spain', 'Malta', 'Taiwan', 'Kuwait', 'Suriname',
       'Trinidad and Tobago', 'El Salvador', 'Guatemala', 'Uzbekistan',
       'Slovakia', 'Japan', 'South Korea', 'Ecuador', 'Bahrain', 'Italy',
       'Bolivia', 'Moldova', 'Paraguay', 'Kazakhstan', 'Slovenia',
       'Lithuania', 'Nicaragua', 'Peru', 'Belarus', 'Poland', 'Malaysia',
       'Croatia', 'Libya', 'Russia', 'Jamaica', 'North Cyprus', 'Cyprus',
       'Algeria', 'Kosovo', 'Turkmenistan', 'Mauritius', 'Hong Kong',
       'Estonia', 'Indonesia', 'Vietnam', 'Turkey', 'Kyrgyzstan',
       'Nigeria', 'Bhutan', 'Azerbaijan', 'Pakistan', 'Jordan',
       'Montenegro', 'China', 'Zambia', 'Romania', 'Serbia', 'Portugal',
       'Latvia', 'Philippines', 'Somaliland region', 'Morocco',
       'Macedonia', 'Mozambique', 'Albania', 'Bosnia and Herzegovina',
       'Lesotho', 'Dominican Republic', 'Laos', 'Mongolia', 'Swaziland',
       'Greece', 'Lebanon', 'Hungary', 'Honduras', 'Tajikistan',
       'Tunisia', 'Palestinian Territories', 'Bangladesh', 'Iran',
       'Ukraine', 'Iraq', 'South Africa', 'Ghana', 'Zimbabwe', 'Liberia',
       'India', 'Sudan', 'Haiti', 'Congo (Kinshasa)', 'Nepal', 'Ethiopia',
       'Sierra Leone', 'Mauritania', 'Kenya', 'Djibouti', 'Armenia',
       'Botswana', 'Myanmar', 'Georgia', 'Malawi', 'Sri Lanka',
       'Cameroon', 'Bulgaria', 'Egypt', 'Yemen', 'Angola', 'Mali',
       'Congo (Brazzaville)', 'Comoros', 'Uganda', 'Senegal', 'Gabon',
       'Niger', 'Cambodia', 'Tanzania', 'Madagascar',
       'Central African Republic', 'Chad', 'Guinea', 'Ivory Coast',
       'Burkina Faso', 'Afghanistan', 'Rwanda', 'Benin', 'Syria',
       'Burundi', 'Togo'], dtype=object)
```

Data Cleaning:-

In [23]:

```
# Checking for the null values:-

hp.isnull().sum()
```

Out[23]:

```
Country                          0
Region                           0
Happiness Rank                   0
Happiness Score                  0
Standard Error                   0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          0
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```

In [24]:

```
hp.dtypes
```

Out[24]:

```
Country                           object
Region                            object
Happiness Rank                     int64
Happiness Score                  float64
Standard Error                   float64
Economy (GDP per Capita)         float64
Family                           float64
Health (Life Expectancy)         float64
Freedom                          float64
Trust (Government Corruption)    float64
Generosity                       float64
Dystopia Residual                float64
dtype: object
```

Since, Country and Region are object(categorical) type ,rest parameters are integer and float type.

In [25]:

```
hp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Country                        158 non-null    object 
 1   Region                         158 non-null    object 
 2   Happiness Rank                 158 non-null    int64  
 3   Happiness Score                158 non-null    float64
 4   Standard Error                 158 non-null    float64
 5   Economy (GDP per Capita)       158 non-null    float64
 6   Family                         158 non-null    float64
 7   Health (Life Expectancy)       158 non-null    float64
 8   Freedom                        158 non-null    float64
 9   Trust (Government Corruption)  158 non-null    float64
 10  Generosity                     158 non-null    float64
 11  Dystopia Residual              158 non-null    float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

In [69]:

```
hp.describe()
```

Out[69]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	

In [59]:

```
# dropping irrelevant columns:-

hp_new=hp.drop(columns=['Country','Happiness Rank','Region'])
```

In [60]:

```
hp_new
```

Out[60]:

	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generos
0	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29
1	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43
2	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34
3	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34
4	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45
...	...	...	...	...	...	...	...	
153	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22
154	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18
155	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47
156	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	0.10062	0.19
157	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	0.10731	0.16

158 rows × 9 columns

In [61]:

```
hp_new.shape
```

Out[61]:

```
(158, 9)
```

In [62]:

```
hp_new.columns
```

Out[62]:

```
Index(['Happiness Score', 'Standard Error', 'Economy (GDP per Capita)',
       'Family', 'Health (Life Expectancy)', 'Freedom',
       'Trust (Government Corruption)', 'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [63]:

```
hp_new.dtypes
```

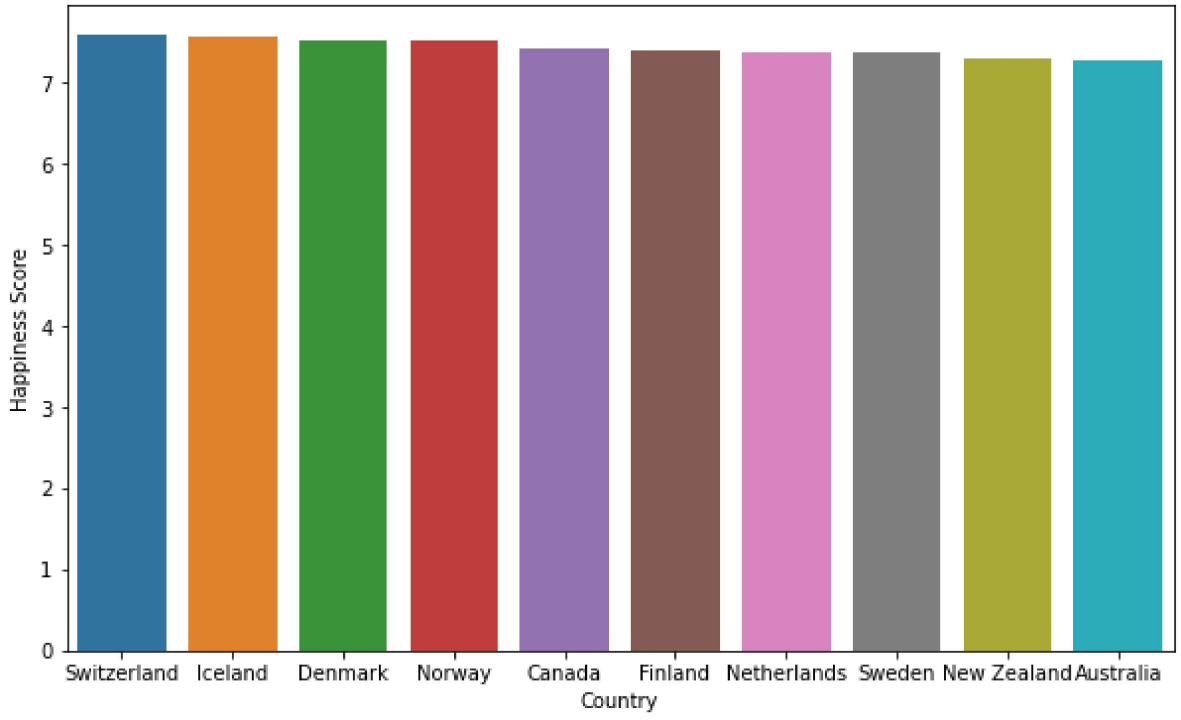
Out[63]:

```
Happiness Score                  float64
Standard Error                   float64
Economy (GDP per Capita)         float64
Family                           float64
Health (Life Expectancy)         float64
Freedom                          float64
Trust (Government Corruption)    float64
Generosity                       float64
Dystopia Residual                float64
dtype: object
```

Now, all he remaining parameters are float type.

In [64]:

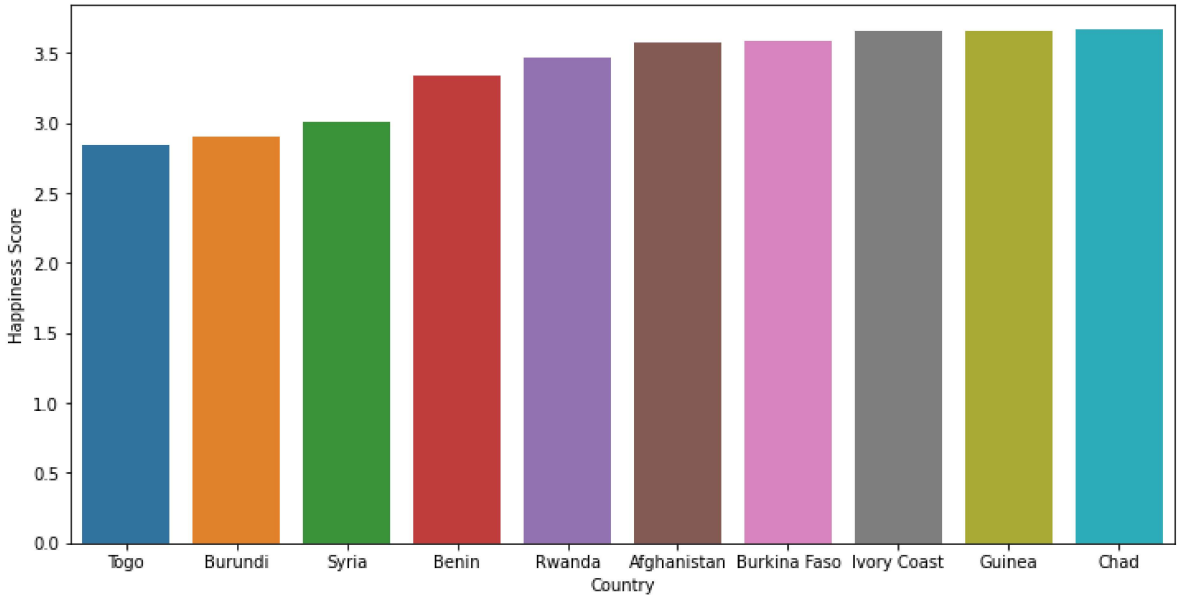
```
# Visualization:-

plt.figure(figsize=(10,6))
hp_happiest=hp.sort_values(by='Happiness Score',ascending=False).iloc[0:10,:]
sns.barplot(x='Country',y='Happiness Score',data=hp_happiest)
plt.show()
```



Above data shows the Countries with Highest Happiness Score.

In [65]:

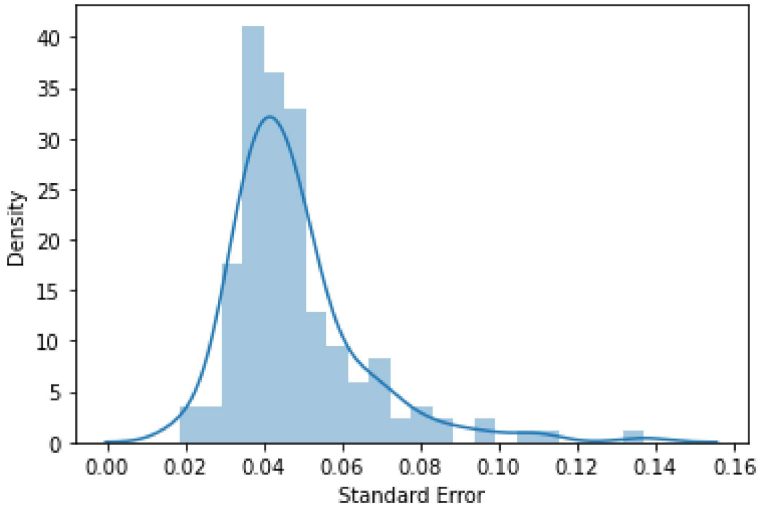
```
plt.figure(figsize=(12,6))
hp_happiest=hp.sort_values(by='Happiness Score',ascending=True).iloc[0:10,:]
sns.barplot(x='Country',y='Happiness Score',data=hp_happiest)
plt.show()
```



Above data shows the Countries with Least Happiness Score.

In [66]:

```
sns.distplot(hp_new['Standard Error'])
plt.show()
```



Above data shows the Distribution of Error in Happiness Score.

In [67]:

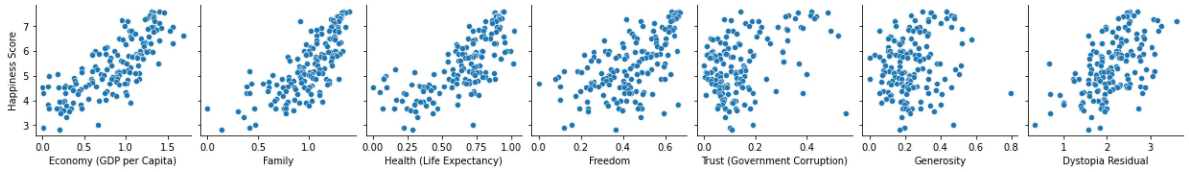
```
hp_new.dtypes
```

Out[67]:

```
Happiness Score                   float64
Standard Error                    float64
Economy (GDP per Capita)          float64
Family                            float64
Health (Life Expectancy)          float64
Freedom                           float64
Trust (Government Corruption)     float64
Generosity                        float64
Dystopia Residual                 float64
dtype: object
```

In [68]:

```
sns.pairplot(data=hp_new,x_vars=['Economy (GDP per Capita)','Family',
                                 'Health (Life Expectancy)','Freedom',
                                'Trust (Government Corruption)','Generosity','Dystopia Resi
                        y_vars='Happiness Score')

plt.show()
```



In [ ]:

```
# From above observations we can see that
#"Economy (GDP per Capita)","Family",
#"Health (Life Expectancy)" have almost
#linear relation with the happiness score.
```

In [80]:

```
# Making a deep copy

hp_updated=hp[['Happiness Score','Economy (GDP per Capita)','Family',
               'Health (Life Expectancy)','Freedom',
               'Trust (Government Corruption)',
               'Generosity','Dystopia Residual']].copy()
```

In [81]:

```
hp_updated
```

Out[81]:

	Happiness Score	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	7.587	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	7.561	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	7.527	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	7.522	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	7.427	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...	...
153	3.465	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	3.340	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	3.006	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858
156	2.905	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302
157	2.839	0.20868	0.13995	0.28443	0.36453	0.10731	0.16681	1.56726

158 rows × 8 columns

In [82]:

```
hp_updated.dtypes
```

Out[82]:

```
Happiness Score                 float64
Economy (GDP per Capita)        float64
Family                          float64
Health (Life Expectancy)        float64
Freedom                         float64
Trust (Government Corruption)   float64
Generosity                      float64
Dystopia Residual               float64
dtype: object
```

In [83]:

```
hp_updated.shape
```

Out[83]:

```
(158, 8)
```

Correlation:-

In [84]:

```
# Checking the Correlation with respect to target variable(happiness score);-

hp_updated.corr()['Happiness Score'].sort_values()
```

Out[84]:

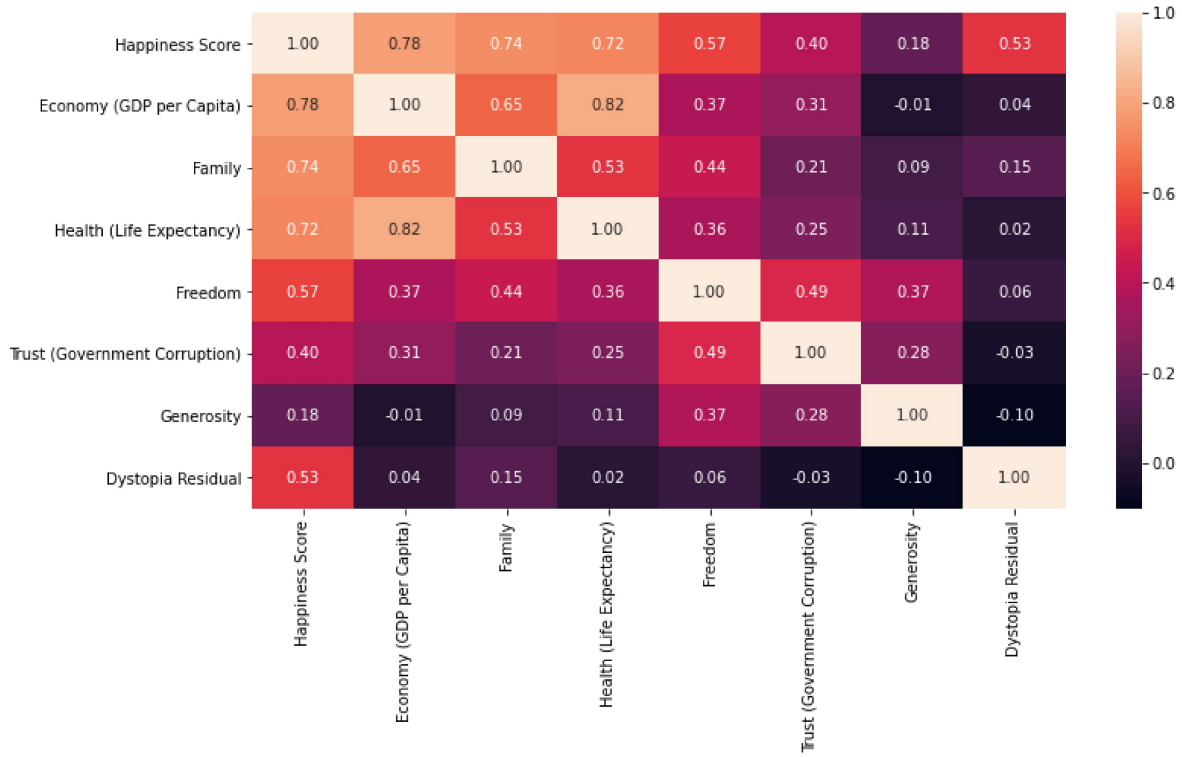
```
Generosity                      0.180319
Trust (Government Corruption)   0.395199
Dystopia Residual               0.530474
Freedom                         0.568211
Health (Life Expectancy)        0.724200
Family                          0.740605
Economy (GDP per Capita)        0.780966
Happiness Score                 1.000000
Name: Happiness Score, dtype: float64
```

In [ ]:

```
#The Above observation shows a good
#correlation with respect to the target variable in which
#Health (Life Expectancy),Family and Economy (GDP per Capita)
#are highly correlated.
```

In [85]:

```
plt.figure(figsize=(12,6))
sns.heatmap(hp_updated.corr(),annot=True,fmt='0.2f')
plt.show()
```



Checking for the Skewness:-

In [86]:

```
hp_updated.skew()
```

Out[86]:

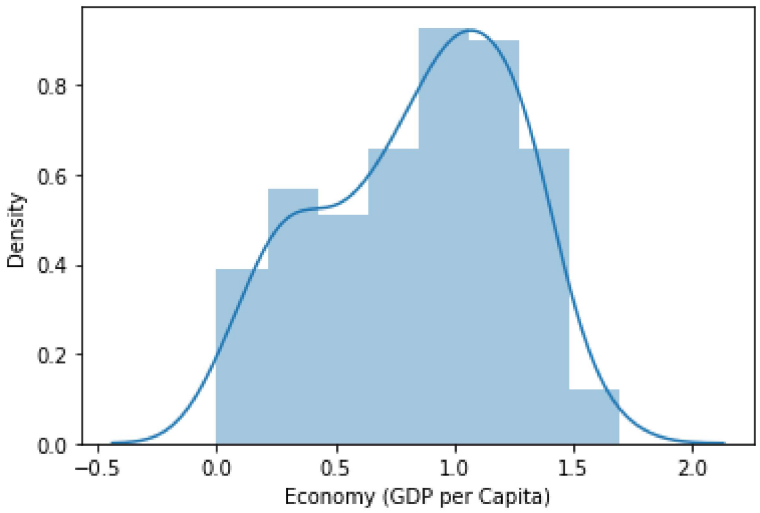
```
Happiness Score                  0.097769
Economy (GDP per Capita)        -0.317575
Family                          -1.006893
Health (Life Expectancy)        -0.705328
Freedom                         -0.413462
Trust (Government Corruption)    1.385463
Generosity                       1.001961
Dystopia Residual               -0.238911
dtype: float64
```

In [ ]:

```
#Above observations shows that 'Family','Trust (Government Corruption)' and
#'Generosity' are highley skewed among others.
```

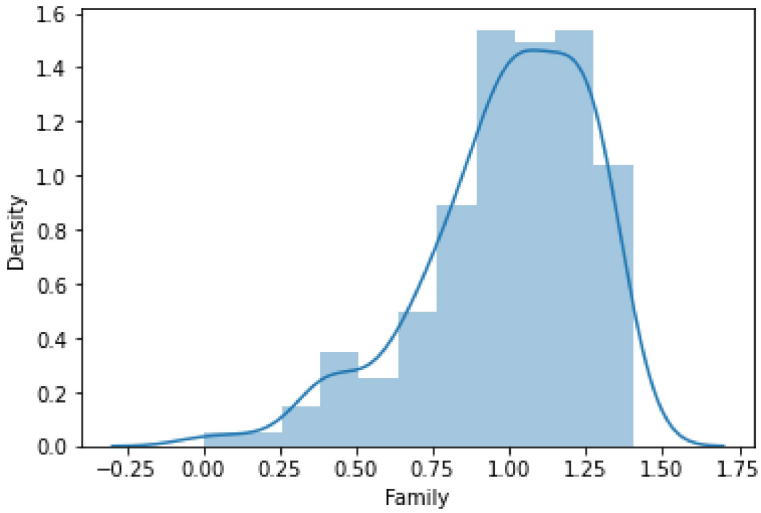
In [89]:

```
# checking with the distribution curve:-

sns.distplot(hp_updated['Economy (GDP per Capita)'])
plt.show()
```



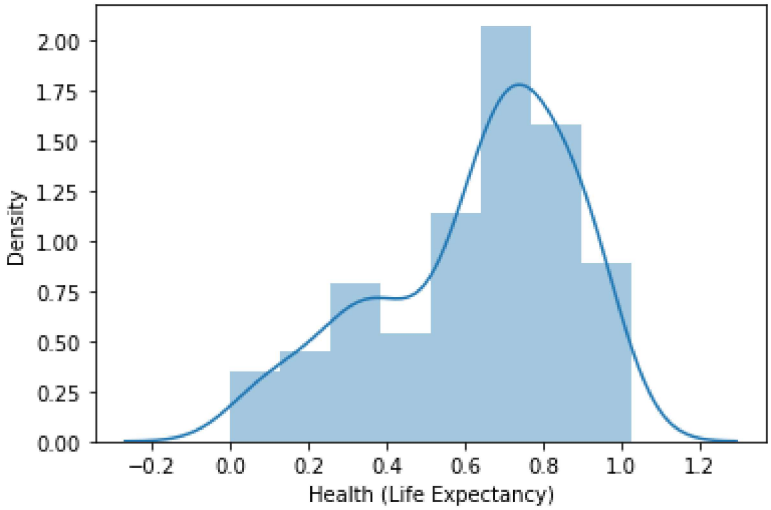
In [90]:

```
sns.distplot(hp_updated['Family'])
plt.show()
```



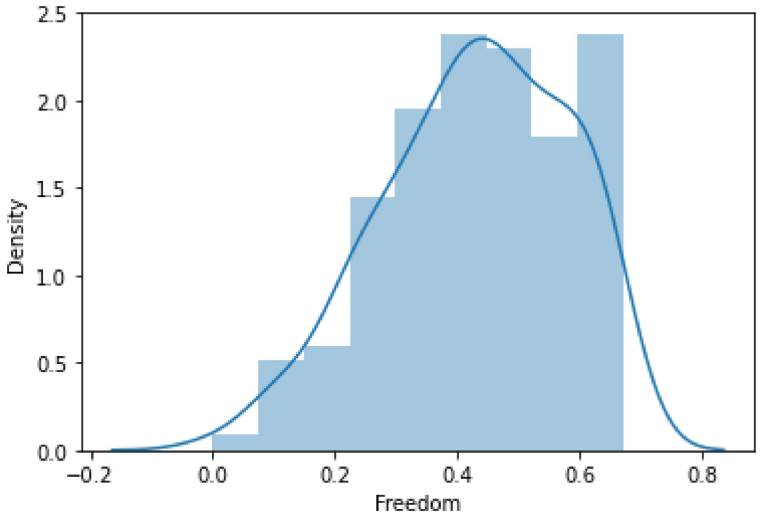
In [91]:

```
sns.distplot(hp_updated['Health (Life Expectancy)'])
plt.show()
```



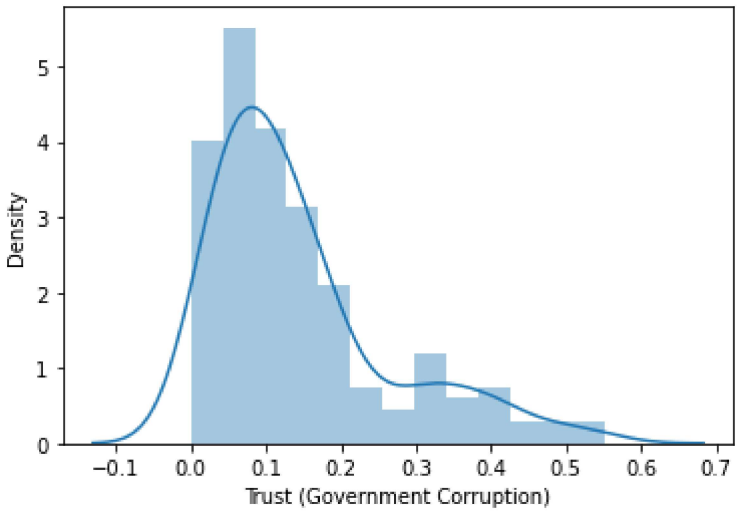
In [92]:

```
sns.distplot(hp_updated['Freedom'])
plt.show()
```



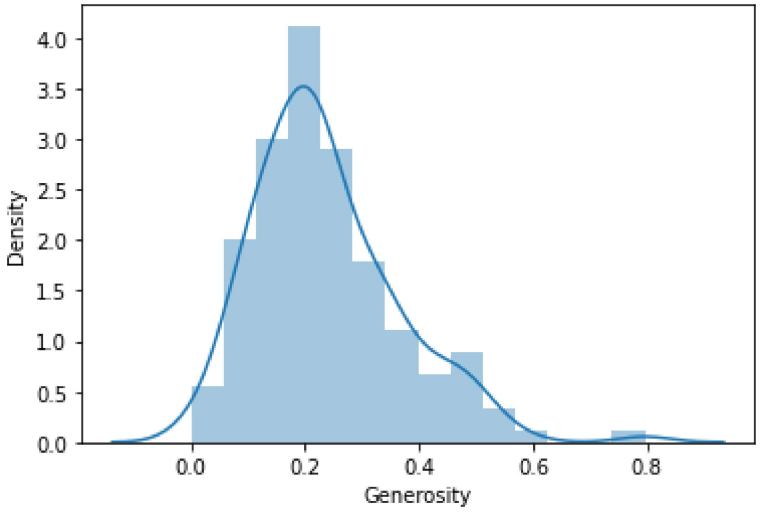
In [93]:

```
sns.distplot(hp_updated['Trust (Government Corruption)'])
plt.show()
```



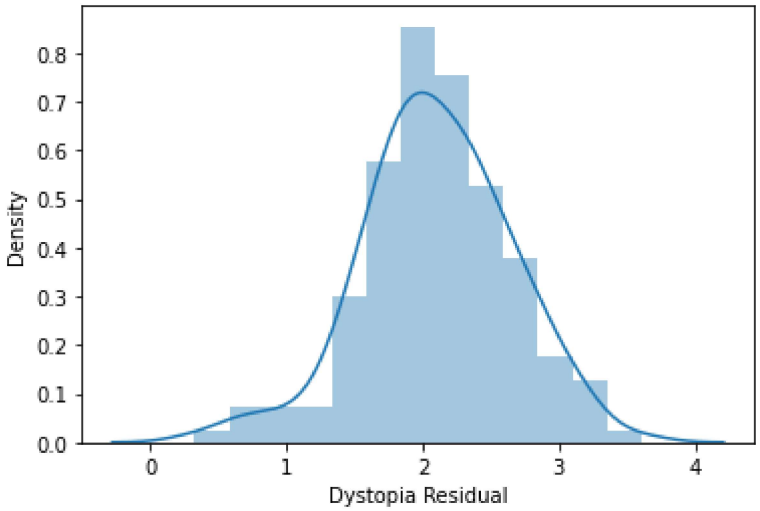
In [94]:

```
sns.distplot(hp_updated['Generosity'])
plt.show()
```



In [95]:

```
sns.distplot(hp_updated['Dystopia Residual'])
plt.show()
```



In [96]:

```
sns.distplot(hp_updated['Happiness Score'])
plt.show()
```

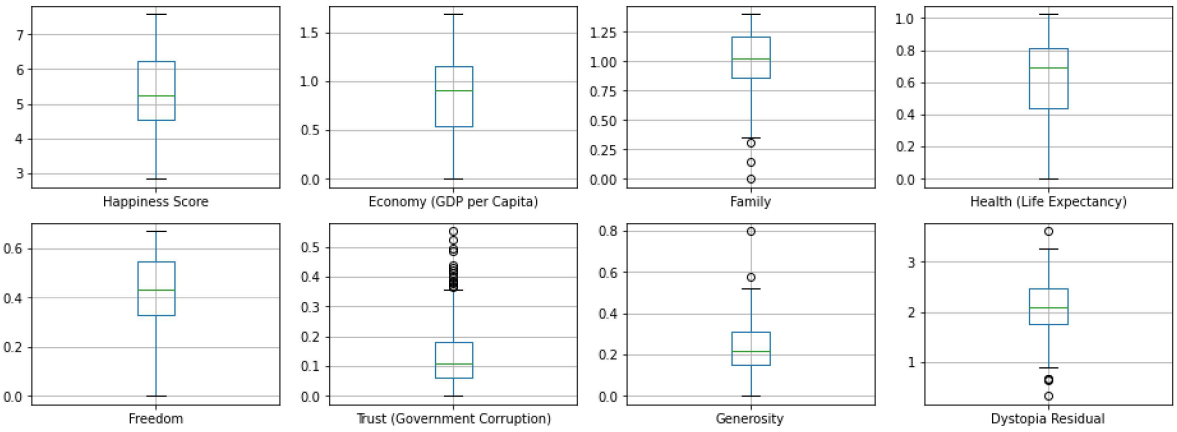
In [ ]:

```
#So, the above visualizations shows that 'Trust(Government Corruption)','Generosity'
#and Dystopia Residual are little skewed and 'Economy(GDP per Capita)','Health','Freedom',
#'Family' and 'Happiness Score' are highly skewed.
#Therefore the Data is not normalized.
```

Outliers Removal:-

In [100]:

```
# checking the outliers with visualization:-

hp_updated.plot(kind='box',subplots=True,layout=(4,4),figsize=(16,12),grid=True)
plt.show()
```



In [101]:

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

Out[101]:

```
array([[1.93736005, 1.36962124, 1.32028142, ..., 2.30965159, 0.47103971,
        0.75825809],
       [1.91458063, 1.13522625, 1.51445776, ..., 0.01647953, 1.57585637,
        1.09285682],
       [1.88479215, 1.19286069, 1.36105403, ..., 2.8427738 , 0.8242928 ,
        0.71233526],
       ...,
       [2.0761992 , 0.45524543, 1.90108634, ..., 0.38141902, 1.85689094,
        3.20843049],
       [2.1646885 , 2.06756644, 2.1184666 , ..., 0.35771452, 0.31694987,
        0.48198451],
       [2.22251319, 1.58633379, 3.13472485, ..., 0.30180313, 0.5581534 ,
        0.96361241]])
```

In [102]:

```
# index no.

np.where(z>3)
```

Out[102]:

```
(array([ 27, 128, 147, 153, 155, 157], dtype=int64),
 array([5, 6, 2, 5, 7, 2], dtype=int64))
```

In [103]:

```
Happiness_new=hp_updated[(z<3).all(axis=1)]
Happiness_new
```

Out[103]:

	Happiness Score	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dysto Resid
0	7.587	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.517
1	7.561	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.702
2	7.527	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.492
3	7.522	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.465
4	7.427	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.451
...	...	...	...	...	...	...	...	
150	3.655	0.46534	0.77115	0.15185	0.46866	0.17922	0.20165	1.417
151	3.587	0.25812	0.85188	0.27125	0.39493	0.12832	0.21747	1.464
152	3.575	0.31982	0.30285	0.30335	0.23414	0.09719	0.36510	1.952
154	3.340	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.633
156	2.905	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.833

152 rows × 8 columns

In [104]:

```
hp_updated.shape
```

Out[104]:

```
(158, 8)
```

In [105]:

```
Happiness_new.shape
```

Out[105]:

```
(152, 8)
```

In [106]:

```
# percentage data loss:-

loss_percent=(158-152)/(158100)
print(loss_percent,'%')
```

```
3.795066413662239e-05 %
```

In [109]:

```
# Transforming the data to remove skewness:-

import sklearn
from sklearn.preprocessing import power_transform
X=power_transform(hp_updated,method='yeo-johnson')
X
```

Out[109]:

```
array([[ 1.85703445,  1.44606101,  1.66920633, ...,  1.77399061,
         0.62239051,  0.75099154],
       [ 1.83666844,  1.17332111,  2.01213244, ...,  0.31599326,
         1.48099498,  1.11001108],
       [ 1.8100018 ,  1.23983557,  1.73958573, ...,  1.90679207,
         0.92797276,  0.70227525],
       ...,
       [-2.20718027, -0.5134688 , -1.69066357, ...,  0.73891461,
         1.65933595, -2.86621557],
       [-2.31217748, -1.89495386, -1.79680304, ..., -0.15194624,
        -0.19482942, -0.51480136],
       [-2.38132666, -1.52122584, -2.16039658, ..., -0.06732623,
        -0.49041465, -0.97664547]])
```

In [110]:

```
X.ndim
```

Out[110]:

```
2
```

In [111]:

```
X.std()
```

Out[111]:

```
1.0
```

In [126]:

```
x=Happiness_new.drop(['Happiness Score'],axis=1)
x
```

Out[126]:

	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...
150	0.46534	0.77115	0.15185	0.46866	0.17922	0.20165	1.41723
151	0.25812	0.85188	0.27125	0.39493	0.12832	0.21747	1.46494
152	0.31982	0.30285	0.30335	0.23414	0.09719	0.36510	1.95210
154	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
156	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302

152 rows × 7 columns

In [127]:

```
y=Happiness_new['Happiness Score']
y
```

Out[127]:

```
0      7.587
1      7.561
2      7.527
3      7.522
4      7.427
       ...
150    3.655
151    3.587
152    3.575
154    3.340
156    2.905
Name: Happiness Score, Length: 152, dtype: float64
```

Preprocessing:-

In [132]:

```
import sklearn
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
```

Machine Learning Algorithms Training and Testing Process:-

In [122]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error as MSE
```

In [133]:

```
# Linear Regression;

lr=LinearRegression()

for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,random_state=i)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    train_accuracy=r2_score(y_train,pred_train)
    test_accuracy=r2_score(y_test,pred_test)
    print("At random state",{i},train_accuracy)
    print("At random state",{i},test_accuracy)
    print("\n")
```

```
At random state {13} 0.9999999367196838
At random state {13} 0.999999929189268


At random state {14} 0.9999999356925638
At random state {14} 0.9999999265301686


At random state {15} 0.9999999376660788
At random state {15} 0.9999999229584069


At random state {16} 0.9999999412459797
At random state {16} 0.9999999008535152


At random state {17} 0.9999999370080925
At random state {17} 0.9999999282857412
```

In [ ]:

```
#Above observations shows that our model is overfitted,lets try
#feature selection.
```

In [145]:

```
x_feature=x.drop(["Economy (GDP per Capita)"],axis=1)
scaler=StandardScaler()
x_feature=scaler.fit_transform(x_feature)
```

In [146]:

```
lr=LinearRegression()

for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x_feature,y,test_size=0.2,random_state=i
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    train_accuracy=r2_score(y_train,pred_train)
    test_accuracy=r2_score(y_test,pred_test)
    print("At random state",{i},train_accuracy)
    print("At random state",{i},test_accuracy)
    print("\n")
```

```
At random state {89} 0.9688166862199774
At random state {89} 0.9676707213939362


At random state {90} 0.9656204756362923
At random state {90} 0.9792083547503251


At random state {91} 0.9720872724144539
At random state {91} 0.9575608377240018


At random state {92} 0.9676264301472125
At random state {92} 0.9748861562761122


At random state {93} 0.9669567539152554
```

In [160]:

```
x_train,x_test,y_train,y_test=train_test_split(x_feature,y,test_size=0.20,random_state=20)
```

In [161]:

```
# Cross Validation:-

train_accuracy=r2_score(y_train,pred_train)
test_accuracy=r2_score(y_test,pred_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(lr,x,y,cv=j)
    cv_mean=cv_score.mean()

    print(j,cv_score,cv_mean,train_accuracy,test_accuracy)
    print("\n")
```

```
2 [0.99999977 0.99999967] 0.999999722557913 -0.8810736515984956 -1.189375044
8130196


3 [0.99999979 0.99999917 0.99999945] 0.999999468361971 -0.8810736515984956 -
1.1893750448130196


4 [0.9999996  0.99999841 0.9999976  0.99999941] 0.9999987556096168 -0.881073
6515984956 -1.1893750448130196


5 [0.99999947 0.99999824 0.99999841 0.99999437 0.99999922] 0.999997944123966
5 -0.8810736515984956 -1.1893750448130196


6 [0.9999991  0.9999991  0.99999708 0.99999408 0.99999369 0.99999905] 0.9999
970173842118 -0.8810736515984956 -1.1893750448130196


7 [0.99999873 0.99999918 0.99998924 0.99999658 0.99999304 0.99999212
 0.99999892] 0.999995402544374 -0.8810736515984956 -1.1893750448130196


8 [0.99999846 0.99999888 0.99999599 0.99999589 0.99999263 0.99998647
 0.99999169 0.99999883] 0.9999948552597964 -0.8810736515984956 -1.1893750448
130196


9 [0.999998   0.99999836 0.99999748 0.99998221 0.99999496 0.99998869
 0.99998401 0.99999624 0.99999877] 0.9999931922183767 -0.8810736515984956 -
1.1893750448130196
```

Hyperparameter Tuning:-

In [162]:

```
# Regularization;

from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [163]:

```
parameters={'alpha':[0.0001,0.001,0.01,0.1,1,10],'random_state':list(range(0,10))}
ls=Lasso()
clf= GridSearchCV(ls,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'alpha': 0.001, 'random_state': 0}
```

In [164]:

```
# final model training;

ls=Lasso(alpha=0.001,random_state=0)
ls.fit(x_train,y_train)
ls_score_training=ls.score(x_train,y_train)
pred_ls=ls.predict(x_test)
ls_score_training*100
```

Out[164]:

```
96.91782537275826
```

In [165]:

```
# Ensemble technique:-

from sklearn.ensemble import RandomForestRegressor
parameters={'criterion':['mse','mae'],'max_features':['auto','sqrt','log2']}
rf= RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'auto'}
```

In [166]:

```
rf=RandomForestRegressor(criterion='mae',max_features='auto')
rf.fit(x_train,y_train)
train_r2=rf.score(x_train,y_train)
pred_decision=rf.predict(x_test)
print(train_r2)
rf2=r2_score(y_test,pred_decision)
print('R2_Score',rf2)
rfs=cross_val_score(rf,x,y,cv=5)
rfc=rf.score(x_train,y_train)
print('Cross_Val_Score',rfc*100)
```

```
0.9861517394931527
R2_Score 0.8934947816125974
Cross_Val_Score 98.61517394931528
```

In [ ]:

```
#So,we are getting our model accuracy and Cross-Validation-Score both as
#98.62% which shows our model is performing
# extremely well".
```

Saving the Model:-

In [167]:

```
import pickle
filename='Happiness_Score.pkl'
pickle.dump(rf,open(filename,'wb'))
```

Conclusion:-

In [168]:

```
loaded_model=pickle.load(open('Happiness_Score.pkl','rb'))
```

In [169]:

```
conclusion=pd.DataFrame([loaded_model.predict(x_test)[:],pred_decision[:]],
                        index=['Predicted','Original'])
conclusion
```

Out[169]:

	0	1	2	3	4	5	6	7	8	
Predicted	4.6189	5.90217	5.0916	4.24937	3.77828	5.07592	4.58626	6.54401	5.01689	7.2492
Original	4.6189	5.90217	5.0916	4.24937	3.77828	5.07592	4.58626	6.54401	5.01689	7.2492

2 rows × 31 columns

In [170]:

```
conclusion.transpose()
```

Out[170]:

	Predicted	Original
0	4.61890	4.61890
1	5.90217	5.90217
2	5.09160	5.09160
3	4.24937	4.24937
4	3.77828	3.77828
5	5.07592	5.07592
6	4.58626	4.58626
7	6.54401	6.54401
8	5.01689	5.01689
9	7.24926	7.24926
10	4.44910	4.44910
11	4.25144	4.25144
12	5.20456	5.20456
13	6.23889	6.23889
14	6.55640	6.55640
15	5.17939	5.17939
16	5.82661	5.82661
17	4.02276	4.02276
18	4.80733	4.80733
19	5.88621	5.88621
20	5.64459	5.64459
21	4.10356	4.10356
22	4.78544	4.78544
23	6.72014	6.72014
24	5.80937	5.80937
25	4.42354	4.42354
26	5.11827	5.11827
27	6.22930	6.22930
28	4.90012	4.90012
29	5.33085	5.33085
30	4.58444	4.58444

In [ ]: