In [1]:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/happiness_score_dataset.csv')
df
```

Out[2]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	0.10731	0.16681	1.56726

158 rows × 12 columns

In [3]:

```
df.isnull().sum()
```

Out[3]:

```
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 [4]:

```
df.columns
```

Out[4]:

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

country and reagion are the categorical data and it also mentioned in problem that hapiness score is the target colum which depands on the economy,family,health,freddom,trust,generosity,dystopia residual

In [5]:

```
df.dtypes
```

Out[5]:

```
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
```

In [6]:

```
df=df.drop(columns=['Country','Region'])
df
```

Out[6]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...	...	...	...
153	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858
156	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302
157	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	0.10731	0.16681	1.56726

158 rows × 10 columns

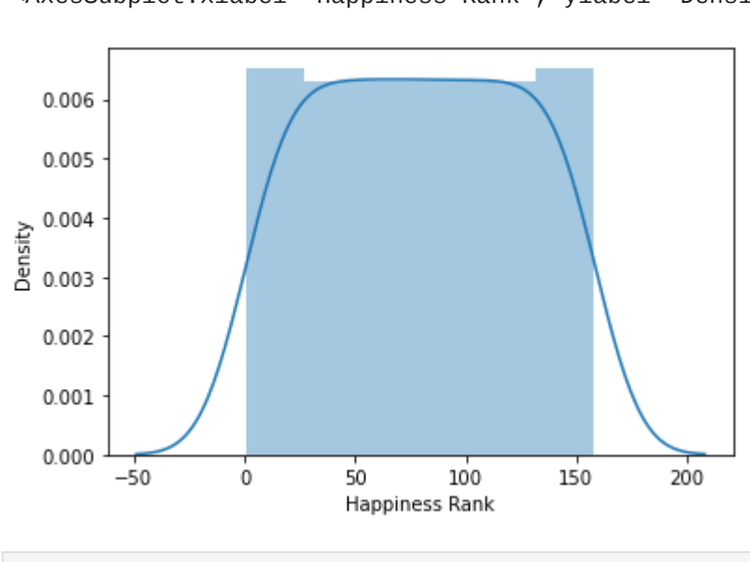
since the target column is not depandent much on the categorical columns so they are removed

In [7]:

```
import seaborn as sns
sns.distplot(df['Happiness Rank'],kde=True)
```

Out[7]:

```
<AxesSubplot:xlabel='Happiness Rank', ylabel='Density'>
```

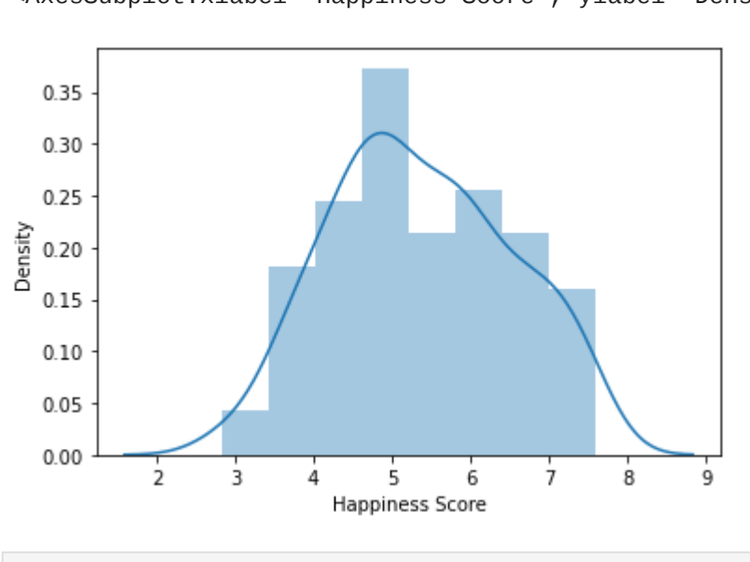


In [8]:

```
sns.distplot(df['Happiness Score'],kde=True)
```

Out[8]:

```
<AxesSubplot:xlabel='Happiness Score', ylabel='Density'>
```

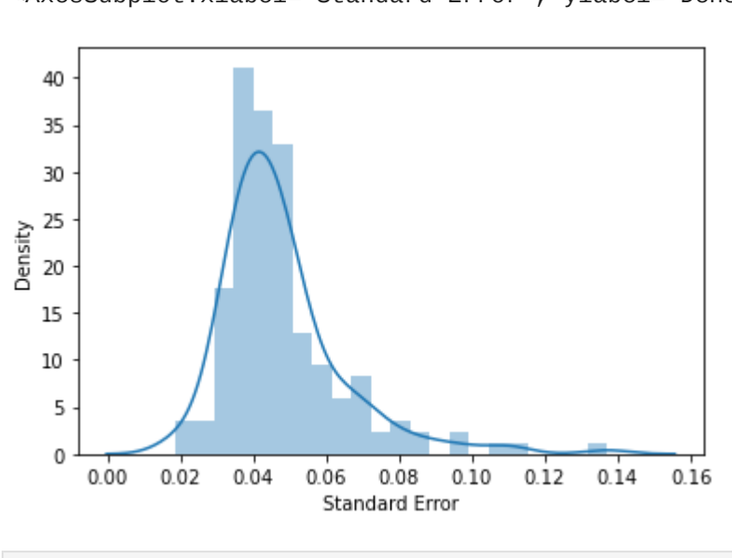


In [9]:

```
sns.distplot(df['Standard Error'],kde=True)
```

Out[9]:

```
<AxesSubplot:xlabel='Standard Error', ylabel='Density'>
```

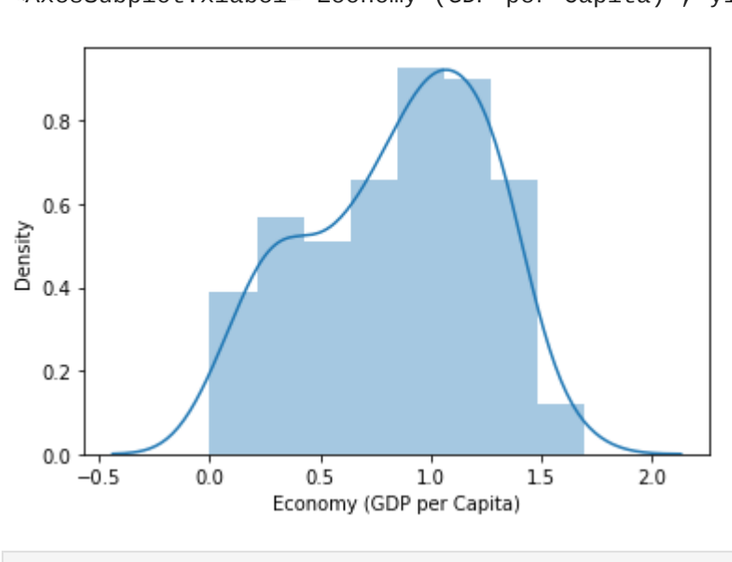


In [10]:

```
sns.distplot(df['Economy (GDP per Capita)'],kde=True)
```

Out[10]:

```
<AxesSubplot:xlabel='Economy (GDP per Capita)', ylabel='Density'>
```

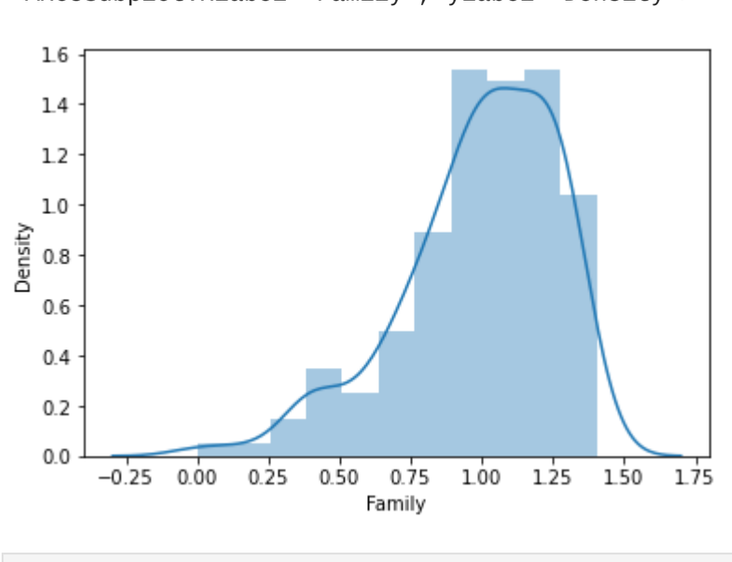


In [11]:

```
sns.distplot(df['Family'],kde=True)
```

Out[11]:

```
<AxesSubplot:xlabel='Family', ylabel='Density'>
```

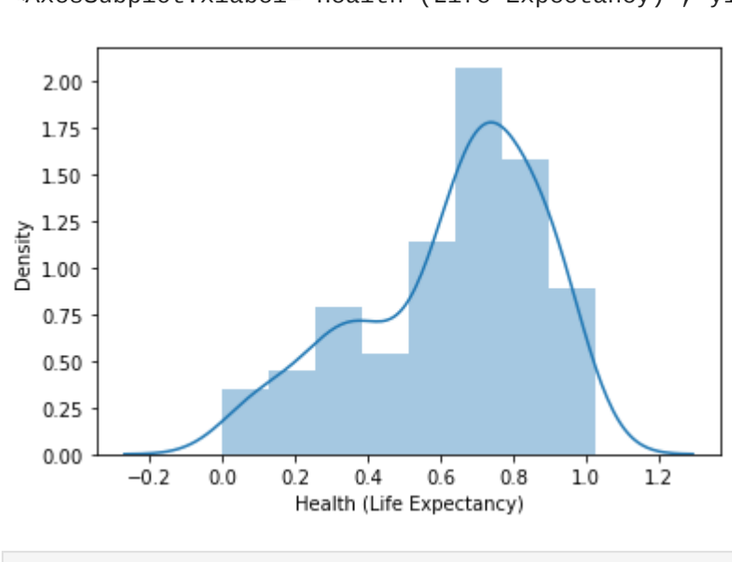


In [12]:

```
sns.distplot(df['Health (Life Expectancy)'],kde=True)
```

Out[12]:

```
<AxesSubplot:xlabel='Health (Life Expectancy)', ylabel='Density'>
```

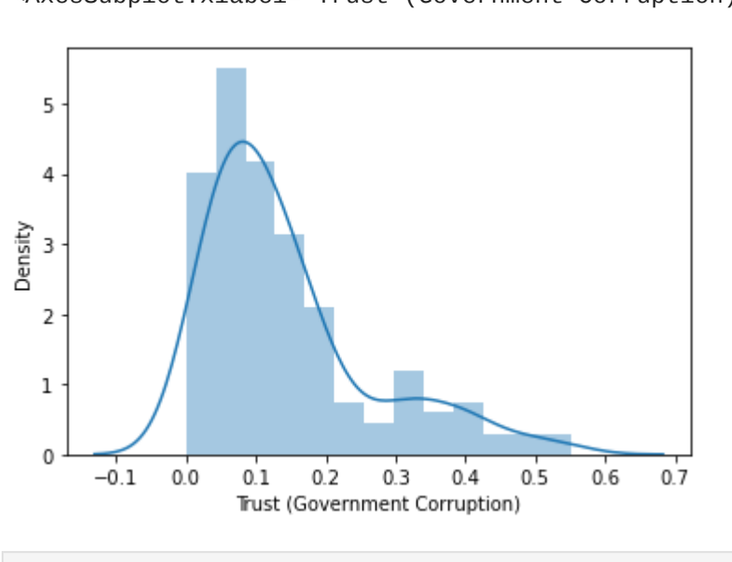


In [13]:

```
sns.distplot(df['Trust (Government Corruption)'],kde=True)
```

Out[13]:

```
<AxesSubplot:xlabel='Trust (Government Corruption)', ylabel='Density'>
```

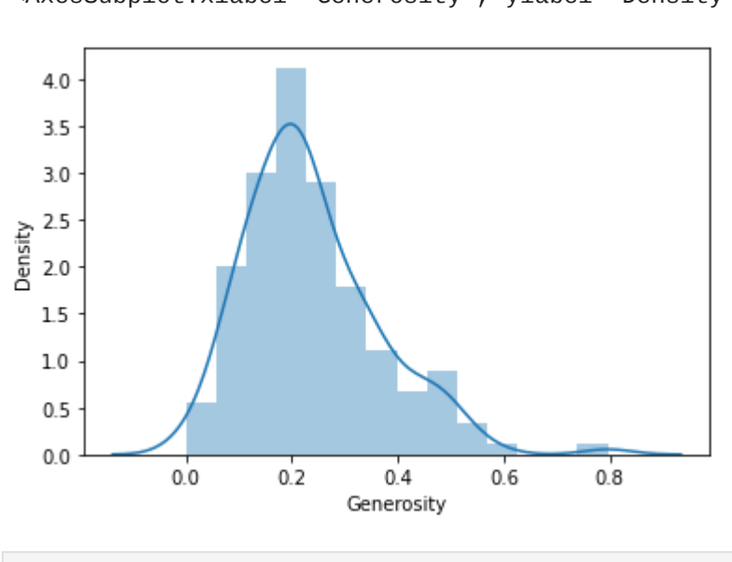


In [14]:

```
sns.distplot(df['Generosity'],kde=True)
```

Out[14]:

```
<AxesSubplot:xlabel='Generosity', ylabel='Density'>
```

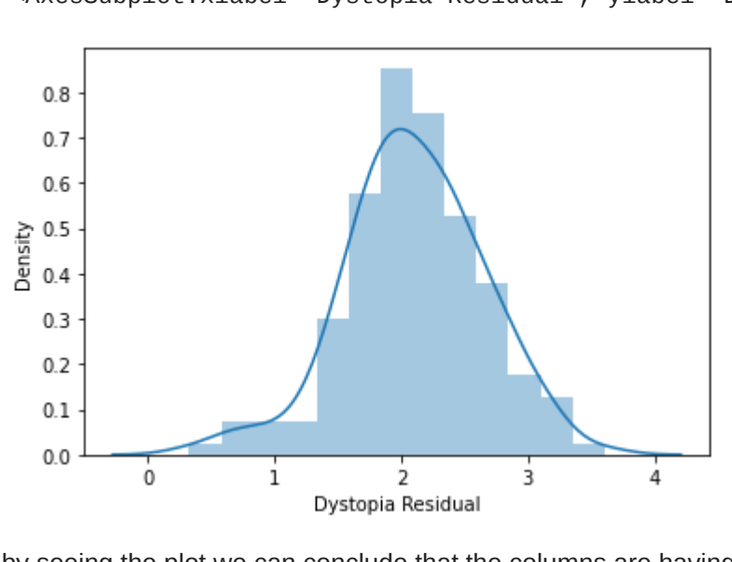


In [15]:

```
sns.distplot(df['Dystopia Residual'],kde=True)
```

Out[15]:

```
<AxesSubplot:xlabel='Dystopia Residual', ylabel='Density'>
```



by seeing the plot we can conclude that the columns are having the skewness

In [16]:

```
df.describe()
```

Out[16]:

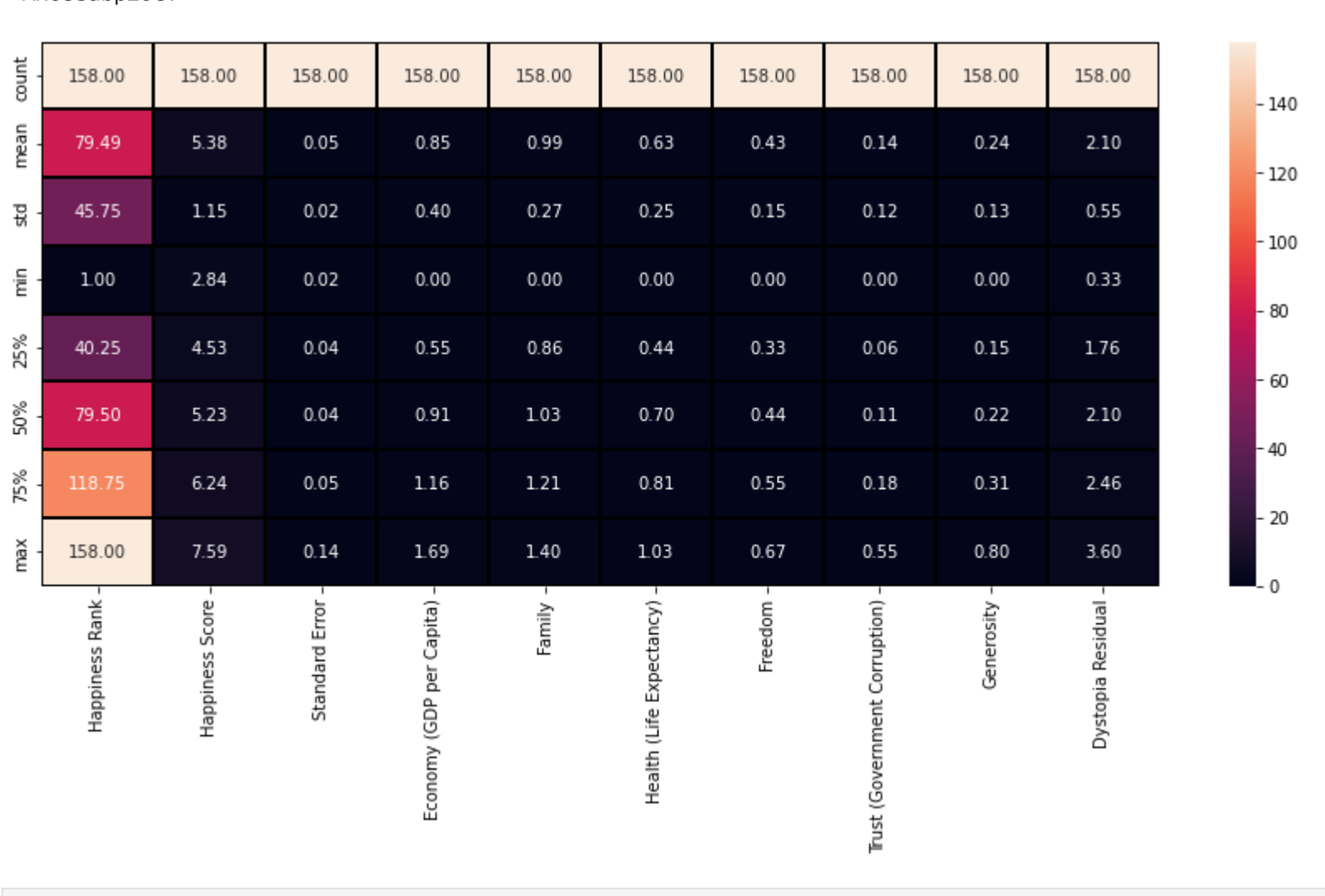
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237296	2.098977
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126685	0.553550
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.328580
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150553	1.759410
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216130	2.095415
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309883	2.462415
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880	3.602140

In [17]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,6))
sns.heatmap(df.describe(),annot=True,linewidth=0.1,linecolor='black',fmt='0.2f')
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
df.corr()
```

Out[18]:

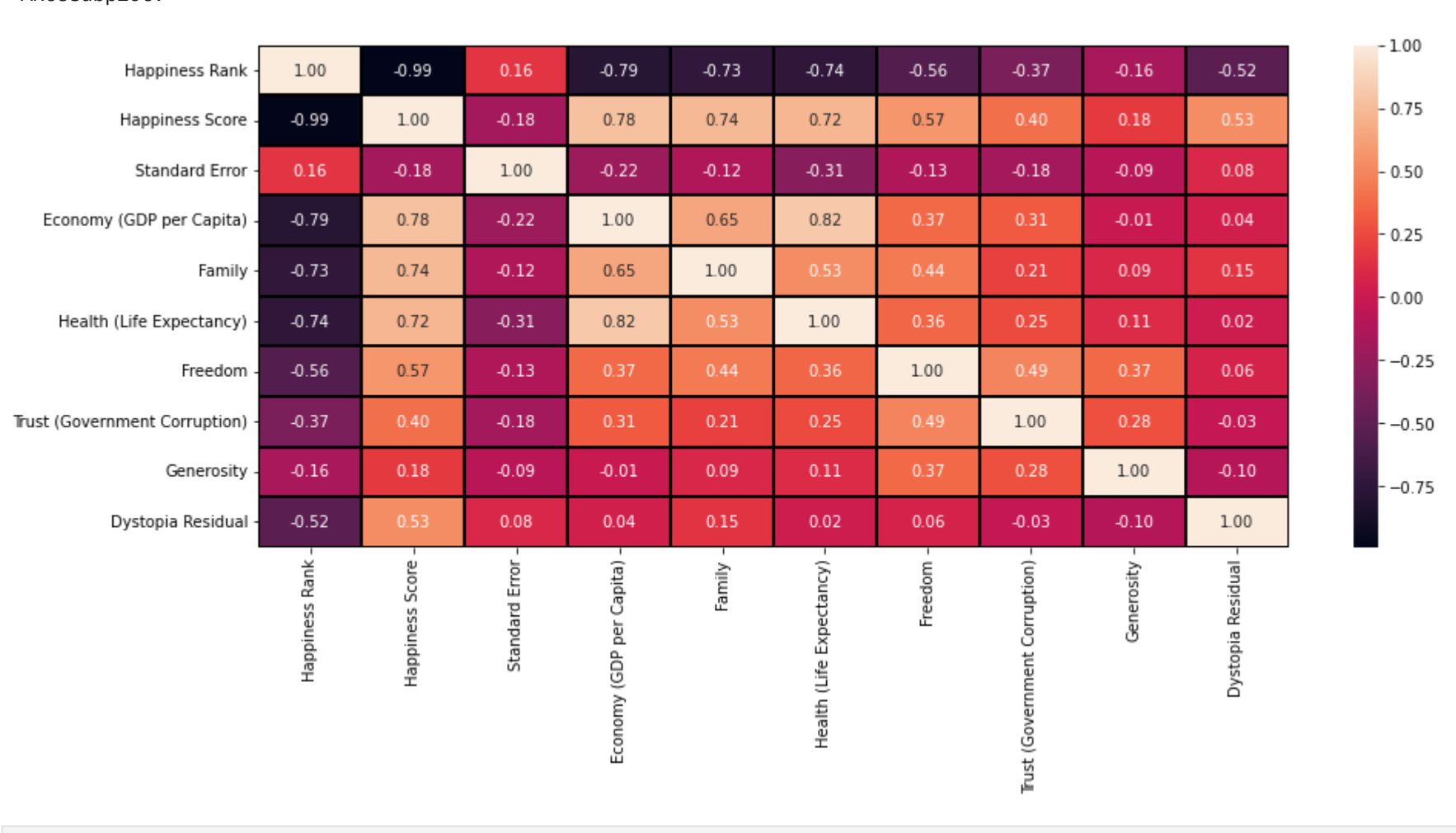
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
Happiness Rank	1.000000	-0.992105	0.158516	-0.785267	-0.733644	-0.735613	-0.556886	-0.372315	-0.160142	-0.521999
Happiness Score	-0.992105	1.000000	-0.177254	0.780966	0.740605	0.724200	0.568211	0.395199	0.180319	0.530474
Standard Error	0.158516	-0.177254	1.000000	-0.217651	-0.120728	-0.310287	-0.129773	-0.178325	-0.088439	0.083981
Economy (GDP per Capita)	-0.785267	0.780966	-0.217651	1.000000	0.645299	0.816478	0.370300	0.307885	-0.010465	0.040059
Family	-0.733644	0.740605	-0.120728	0.645299	1.000000	0.531104	0.441518	0.205605	0.087513	0.148117
Health (Life Expectancy)	-0.735613	0.724200	-0.310287	0.816478	0.531104	1.000000	0.360477	0.248335	0.108335	0.018979
Freedom	-0.556886	0.568211	-0.129773	0.370300	0.441518	0.360477	1.000000	0.493524	0.373916	0.062783
Trust (Government Corruption)	-0.372315	0.395199	-0.178325	0.307885	0.205605	0.248335	0.493524	1.000000	0.276123	-0.033105
Generosity	-0.160142	0.180319	-0.088439	-0.010465	0.087513	0.108335	0.373916	0.276123	1.000000	-0.101301
Dystopia Residual	-0.521999	0.530474	0.083981	0.040059	0.148117	0.018979	0.062783	-0.033105	-0.101301	1.000000

In [19]:

```
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(),annot=True,linewidth=0.1,linecolor='black',fmt='0.2f')
```

Out[19]:

```
<AxesSubplot:>
```



In [20]:

```
df.skew()
```

Out[20]:

```
Happiness Rank                   0.000418
Happiness Score                  0.097769
Standard Error                   1.983439
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
```

columns like Standard Error,Family,Health (Life Expectancy),Trust (Government Corruption),Generosity, Dystopia Residual having the skewness

In [21]:

```
df
```

Out[21]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...	...	...	...
153	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858
156	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302
157	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	0.10731	0.16681	1.56726

158 rows × 10 columns

In [22]:

```
columns_list=df.columns.values
ncol=30
nrows=15
plt.figure(figsize=(ncol,ncol))
for i in range (0,len(columns_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df[columns_list[i]],color='red',orient='h')
    plt.tight_layout()
```

by seeing the box plot we can concluse that the data set having the outliers

In [23]:

```
from scipy.stats import zscore
z=np.abs(zscore(df))
print(df.shape)
print(z.shape)
threshold=3
print(np.where(z>3))
len(np.where(z>3)[0])
```

```
(158, 10)
(158, 10)
(array([ 27,  40,  64, 115, 128, 147, 153, 155, 157], dtype=int64), array([7, 2, 2, 2, 8, 4, 7, 9, 4], dtype=int64))
```

Out[23]:

```
9
```

In [24]:

```
df_new=df[(z<3).all(axis=1)]
print(df_new.shape)
```

```
(149, 10)
```

all the outliers are removed

In [25]:

```
data_loss=((158-149)/158)*100
data_loss
```

Out[25]:

```
5.69620253164557
```

In [26]:

```
x=df_new.iloc[:,:-1]
y=df_new.iloc[:,-1]
print(x)
print(y)
```

```
     Happiness Rank  Happiness Score  Standard Error  \
0                 1            7.587         0.03411
1                 2            7.561         0.04884
2                 3            7.527         0.03328
3                 4            7.522         0.03880
4                 5            7.427         0.03553
..              ...              ...             ...
150             151            3.655         0.05141
151             152            3.587         0.04324
152             153            3.575         0.03084
154             155            3.340         0.03656
156             157            2.905         0.08658

     Economy (GDP per Capita)   Family  Health (Life Expectancy)  Freedom  \
0                     1.39651  1.34951                   0.94143  0.66557
1                     1.30232  1.40223                   0.94784  0.62877
2                     1.32548  1.36058                   0.87464  0.64938
3                     1.45900  1.33095                   0.88521  0.66973
4                     1.32629  1.32261                   0.90563  0.63297
..                        ...      ...                       ...      ...
150                   0.46534  0.77115                   0.15185  0.46866
151                   0.25812  0.85188                   0.27125  0.39493
152                   0.31982  0.30285                   0.30335  0.23414
154                   0.28665  0.35386                   0.31910  0.48450
156                   0.01530  0.41587                   0.22396  0.11850

     Trust (Government Corruption)  Generosity
0                          0.41978     0.29678
1                          0.14145     0.43630
2                          0.48357     0.34139
3                          0.36503     0.34699
4                          0.32957     0.45811
..                             ...         ...
150                        0.17922     0.20165
151                        0.12832     0.21747
152                        0.09719     0.36510
154                        0.08010     0.18260
156                        0.10062     0.19727

[149 rows x 9 columns]
0      2.51738
1      2.70201
2      2.49204
3      2.46531
4      2.45176
        ...
150    1.41723
151    1.46494
152    1.95210
154    1.63328
156    1.83302
Name: Dystopia Residual, Length: 149, dtype: float64
```

In [27]:

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

In [28]:

```
pd.DataFrame(x).skew()
```

Out[28]:

```
0   -0.266578
1   -0.017940
2   -0.031216
3   -0.140644
4   -0.148528
5   -0.190663
6   -0.076489
7    0.174593
8    0.025172
dtype: float64
```

all the columns are in skew range + or - 0.6 so we can consider

In [29]:

```
x
```

Out[29]:

```
array([[-2.05151352,  1.82591163, -0.95033703, ...,  1.75498234,
         1.81762514,  0.65957855],
       [-1.99584442,  1.80625026,  0.37856182, ...,  1.44218195,
         0.32967608,  1.56260401],
       [-1.94524081,  1.7804847 , -1.04140135, ...,  1.6160368 ,
         1.95339957,  0.97722591],
       ...,
       [ 1.53128908, -1.78998689, -1.32081011, ..., -1.26539516,
        -0.19284339,  1.13425805],
       [ 1.56626337, -2.05390645, -0.69272644, ...,  0.31812329,
        -0.4344978 , -0.31207994],
       [ 1.60111233, -2.56623507,  2.17876604, ..., -1.84989335,
        -0.14727363, -0.17248608]])
```

In [30]:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [31]:

```
mms.fit_transform(x)
```

Out[31]:

```
array([[0.        , 1.        , 0.38024301, ..., 0.99131621, 0.96175317,
        0.66907512],
       [0.01524085, 0.99552352, 0.62132033, ..., 0.9159493 , 0.58524637,
        0.8568861 ],
       [0.02909488, 0.98965724, 0.36372291, ..., 0.95783832, 0.99610919,
        0.73513933],
       ...,
       [0.98088409, 0.17673549, 0.31303499, ..., 0.26357885, 0.45302938,
        0.76779883],
       [0.9904592 , 0.11664652, 0.42697649, ..., 0.64511577, 0.39188177,
        0.46698987],
       [1.        , 0.        , 0.94789778, ..., 0.12274838, 0.46456024,
        0.49602256]])
```

In [32]:

```
for i in range(0,1000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=i,test_size=0.2)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test, pred_test)*100,1):
        print('at random state',i,'the model performs very well')
        print('at random state = ',i)
        print('tarining accurecy score = ',r2_score(y_train,pred_train)*100)
        print('testing accurecy score = ',r2_score(y_test,pred_test)*100)
```

```
at random state 24 the model performs very well
at random state =  24
tarining accurecy score =  96.88431737466819
testing accurecy score =  96.89887960298267
at random state 95 the model performs very well
at random state =  95
tarining accurecy score =  96.88723453077314
testing accurecy score =  96.88918050418516
at random state 104 the model performs very well
at random state =  104
tarining accurecy score =  96.945666239193
testing accurecy score =  96.91428838503893
at random state 180 the model performs very well
at random state =  180
tarining accurecy score =  96.9321755200436
testing accurecy score =  96.9356541103891
at random state 355 the model performs very well
at random state =  355
tarining accurecy score =  96.93995494527536
testing accurecy score =  96.91388090861305
at random state 359 the model performs very well
at random state =  359
tarining accurecy score =  96.94355090200122
testing accurecy score =  96.86527726676657
at random state 445 the model performs very well
at random state =  445
tarining accurecy score =  96.90020274775341
testing accurecy score =  96.91591109497566
at random state 520 the model performs very well
at random state =  520
tarining accurecy score =  96.87523588852025
testing accurecy score =  96.86494554708598
at random state 522 the model performs very well
at random state =  522
tarining accurecy score =  96.90968114967743
testing accurecy score =  96.93740316353079
at random state 596 the model performs very well
at random state =  596
tarining accurecy score =  96.95239833667684
testing accurecy score =  96.97010097077552
at random state 639 the model performs very well
at random state =  639
tarining accurecy score =  96.90985302720459
testing accurecy score =  96.85950006044956
at random state 671 the model performs very well
at random state =  671
tarining accurecy score =  96.95928086770891
testing accurecy score =  96.95682770363813
at random state 673 the model performs very well
at random state =  673
tarining accurecy score =  96.86280318682175
testing accurecy score =  96.87079028787343
at random state 703 the model performs very well
at random state =  703
tarining accurecy score =  96.84016035724191
testing accurecy score =  96.8236121693594
at random state 751 the model performs very well
at random state =  751
tarining accurecy score =  96.81177001197028
testing accurecy score =  96.78681640730571
at random state 752 the model performs very well
at random state =  752
tarining accurecy score =  96.8982979458324
testing accurecy score =  96.90113839671527
at random state 786 the model performs very well
at random state =  786
tarining accurecy score =  96.94354560010667
testing accurecy score =  96.93279265904306
at random state 791 the model performs very well
at random state =  791
tarining accurecy score =  96.94041310640722
testing accurecy score =  96.91210541962182
at random state 839 the model performs very well
at random state =  839
tarining accurecy score =  96.92430449530731
testing accurecy score =  96.91420786912978
at random state 864 the model performs very well
at random state =  864
tarining accurecy score =  96.91779998493655
testing accurecy score =  96.93640741161273
at random state 869 the model performs very well
at random state =  869
tarining accurecy score =  96.95080495219563
testing accurecy score =  96.96095164455126
```

In [33]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=869)
lr.fit(x_train,y_train)
pred_y=lr.predict(x_test)
print("r2_score= ",r2_score(y_test,pred_y)*100)
```

```
r2_score=  96.96095164455126
```

In [34]:

```
train_accuracy=(r2_score(y_train,pred_train))
test_accuracy=(r2_score(y_test,pred_test))
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(lr,x,y,cv=5)
cv_mean=cv_score.mean()
print("the cv score is %s and accuracy score for training is %s  and accuracy score for testing is %s "%(cv_mean,train_accuracy,test_accuracy))
```

```
the cv score is 0.8114899818010608 and accuracy score for training is -0.8340032558084522  and accuracy score for testing is -1.1309038536615392
```

In [45]:

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision=rf.predict(x_test)
rfs=r2_score(y_test,pred_decision)
print('R2_score=',rfs*100)
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(rf,x,y,cv=3)
cv_mean=rfscore.mean()
print("cross_val_score=",cv_mean*100)
```

```
R2_score= 67.13345236727979
cross_val_score= -88.77803662053284
```

linear regressor have the good R2_score value with Good cv_score value so model is performing well and giving the good results for linear regresson (R2_score=96.96 and CV_score=81.14)

In [ ]: