In [ ]:

```
```

In [2]:

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

In [3]:

```
s=pd.read_csv(r"C:\Users\user\Downloads\2015 - 2015.csv")
s
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Gc C
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	
...	...	...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	

158 rows × 12 columns

In [4]:
```
s.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 [5]:
```
# to display summary of the statistic
s.describe()
```

Out[5]:

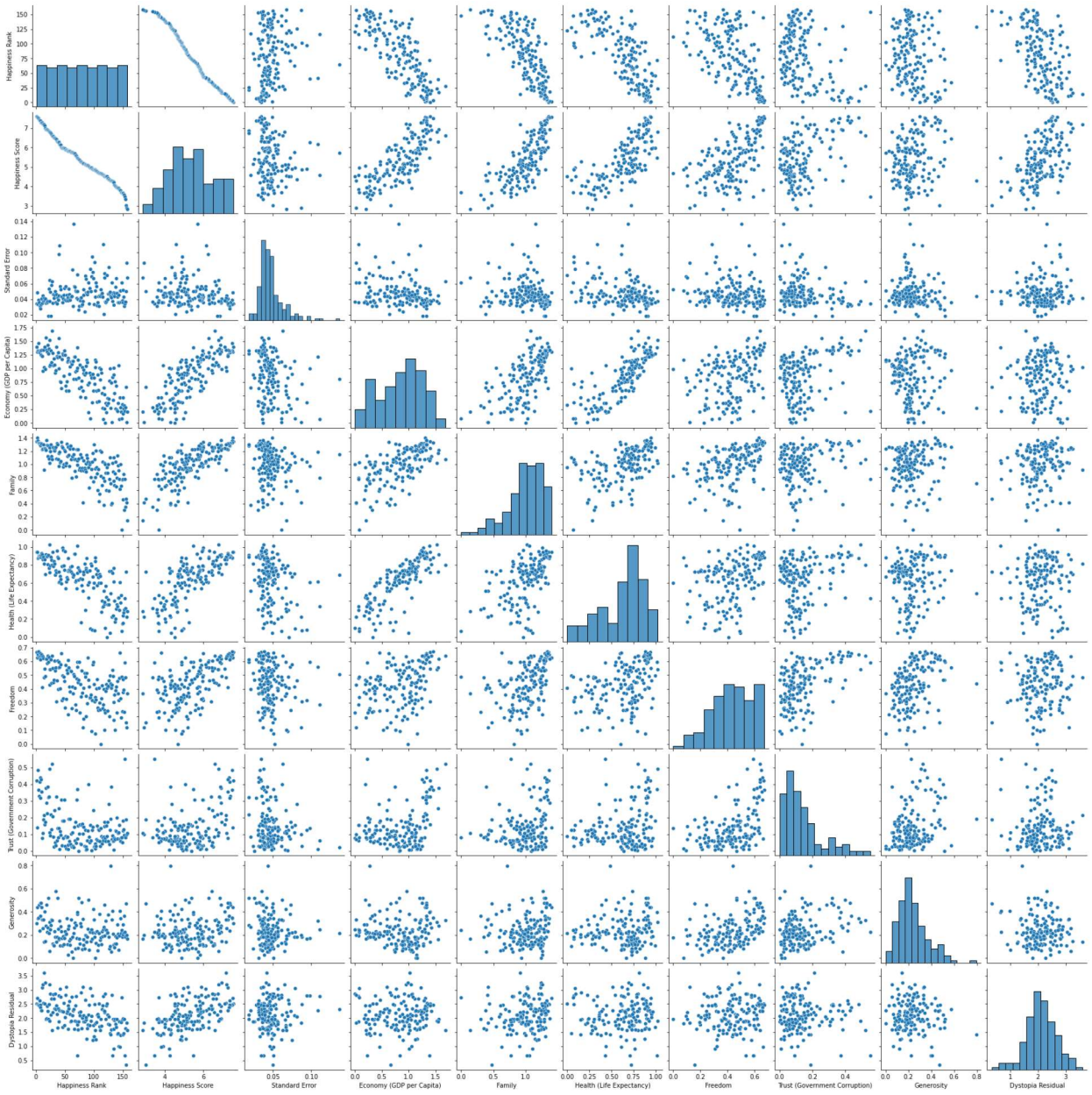
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615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

In [6]:
```
s.columns
```

Out[6]:
```
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')
```

In [7]:
```
sns.pairplot(s)
```

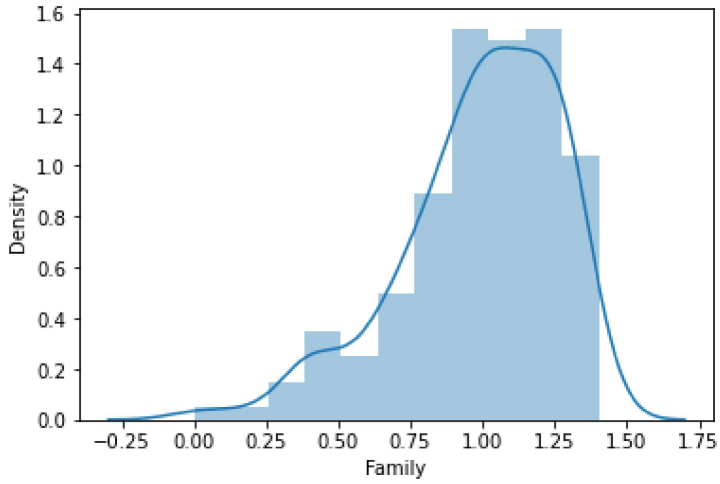
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x2145527eb50>
```



In [8]:
```
sns.distplot(s['Family'])
```

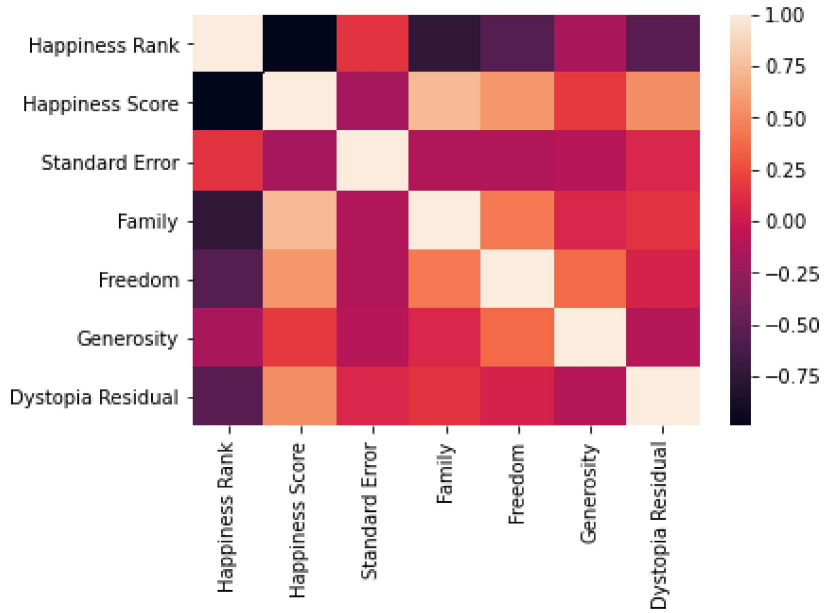
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar flex
ibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]: <AxesSubplot:xlabel='Family', ylabel='Density'>



In [9]:
```
s1=s[['Happiness Rank','Happiness Score','Standard Error','Family','Freedom','Generosi
sns.heatmap(s1.corr())
```

Out[9]: <AxesSubplot:>



In [10]:
```
x=s1[['Happiness Rank','Happiness Score','Standard Error','Family','Freedom','Generosi
y=s1['Dystopia Residual']
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [12]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[12]:
```
LinearRegression()
```

In [13]:
```
lr.intercept_
```

Out[13]:
```
-4.847933341869336
```

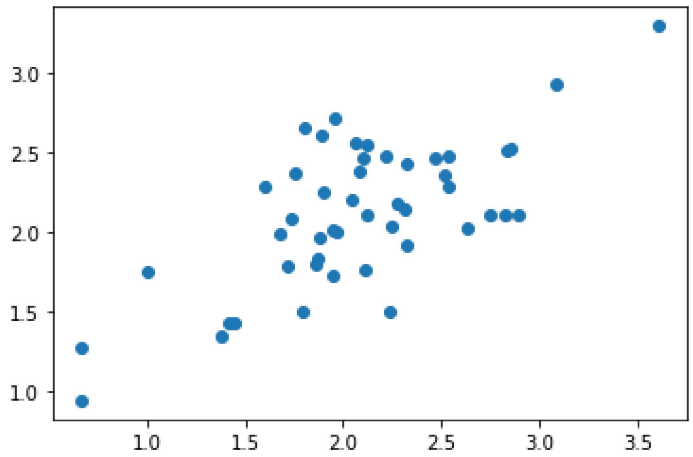
In [14]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[14]:

	Co-efficient
Happiness Rank	0.019180
Happiness Score	1.325506
Standard Error	5.346752
Family	-1.120689
Freedom	-1.523109
Generosity	-0.779344

In [15]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x2145bcf7a90>
```



In [16]:
```
print(lr.score(x_test,y_test))
```

```
0.46003349600597077
```

In [17]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [18]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [19]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
```

Out[19]: 0.4980286962945414

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_test,y_test)
```

Out[20]: 0.05453255748535302

In [21]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[21]: ElasticNet()

In [22]:
```
print(en.coef_)
```

```
[-0.00504537  0.          0.         -0.         -0.         -0.        ]
```

In [23]:
```
print(en.intercept_)
```

```
2.4954729644823126
```

In [24]:
```
print(en.predict(x_test))
```

```
[1.85471055 1.92534577 2.2230228  2.11202459 1.71848546 1.77902994
 2.30374878 1.98084488 2.38952012 2.06661623 2.17256907 1.93039115
 1.95057264 1.74371233 1.77398457 2.374384   1.82948368 2.36933863
 2.10193384 2.01616249 2.43997386 2.19275056 1.79416607 2.25834042
 2.03634399 1.9203004  1.95561801 2.08175235 2.42988311 2.46015535
 2.42483774 1.73362158 2.01111712 1.94552727 2.27852191 1.75884845
 1.79921144 2.25329504 2.08175235 2.0716616  2.23815892 2.27347654
 1.86984667 2.20788668 2.24824967 2.34411176 1.81434756 1.72353084]
```

In [25]:
```
print(en.score(x_test,y_test))
```

```
0.4053166456037285
```

In [26]:
```
from sklearn import metrics
```

In [27]:
```
print("Mean Absolute Error",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error 0.32530238297960257
```

In [28]:
```
print("Mean squared Error",metrics.mean_squared_error(y_test,prediction))
```

```
Mean squared Error 0.17033791032433773
```

In [29]:
```
print("Root Mean squared Error",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean squared Error 0.4127201355935251
```

In [ ]:
```

```