In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as py
import seaborn as sns
```

In [2]:
```
d=pd.read_csv(r"C:\Users\user\Downloads\2015 - 2015.csv")
d
```

Out[2]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom
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 [3]:
```
d.head()
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Go Co
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	

In [4]:

```
d.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]:

```
d.describe()
```

Out[5]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trus (Governmer Corruptior
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.00000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.14342
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.12003
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.00000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.06167
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.10722
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.18025

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trus (Governmer Corruptior
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.55191

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

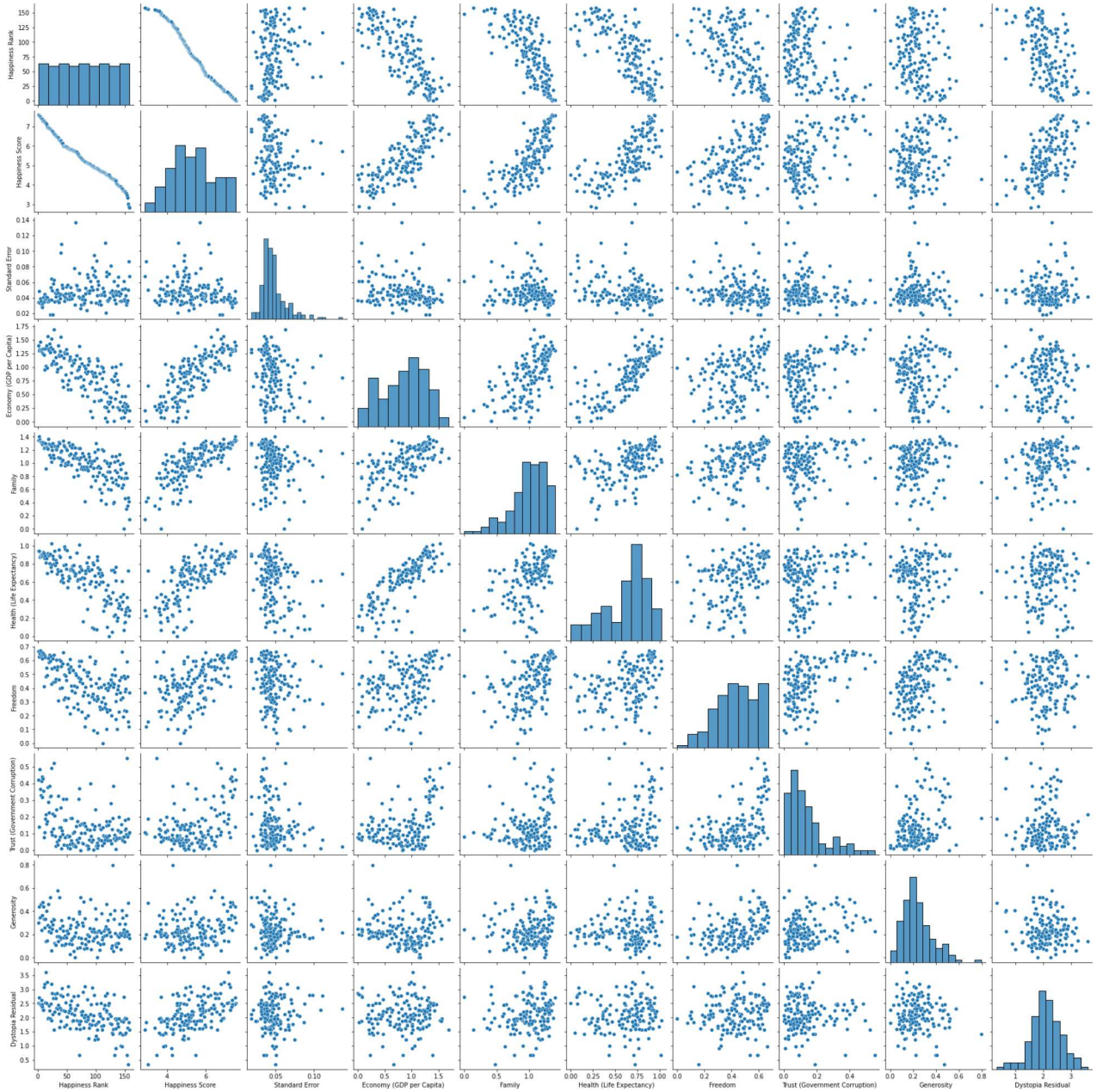
Out[6]:
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [7]:
```
d.index
```

Out[7]:
```
RangeIndex(start=0, stop=158, step=1)
```

In [8]:
```
sns.pairplot(d)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1ff954da880>
```



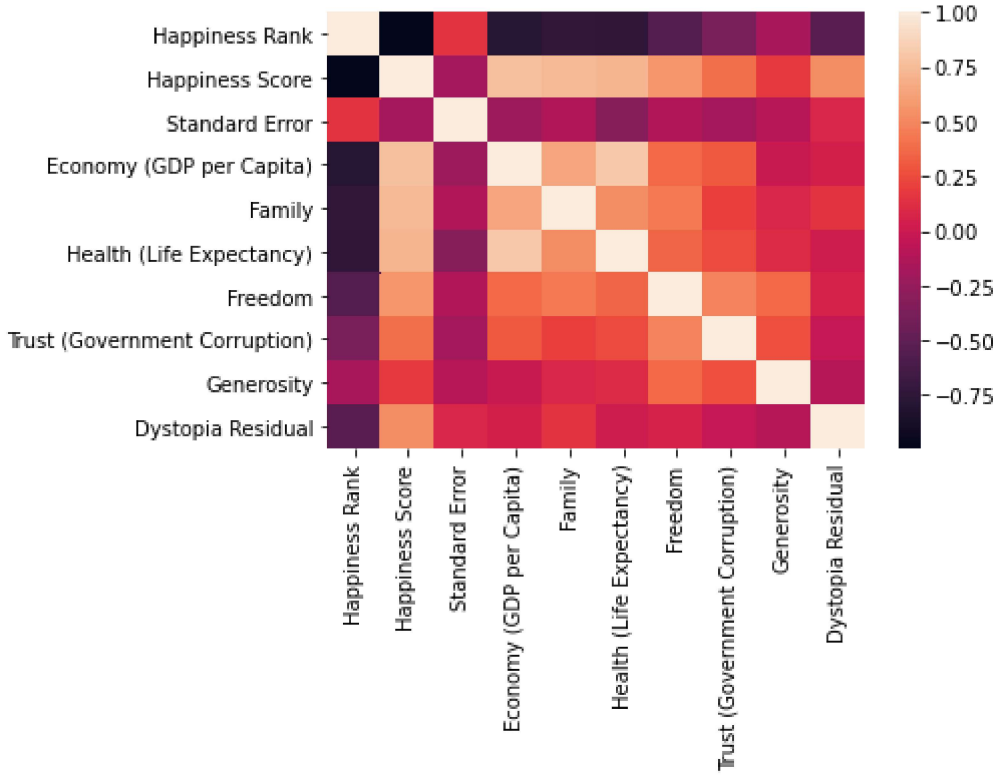
In [9]:
```
sns.distplot(d['Happiness Score'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]: <AxesSubplot:xlabel='Happiness Score', ylabel='Density'>

In [10]:
```
d1=d[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
sns.heatmap(d1.corr())
```

Out[10]: <AxesSubplot:>



In [11]:
```
x=d1[['Happiness Rank','Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y=d1['Happiness Score']
```

In [12]:
```
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 [13]:
```
from sklearn.linear_model import LinearRegression
```

In [14]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
0.002062874964623873
```

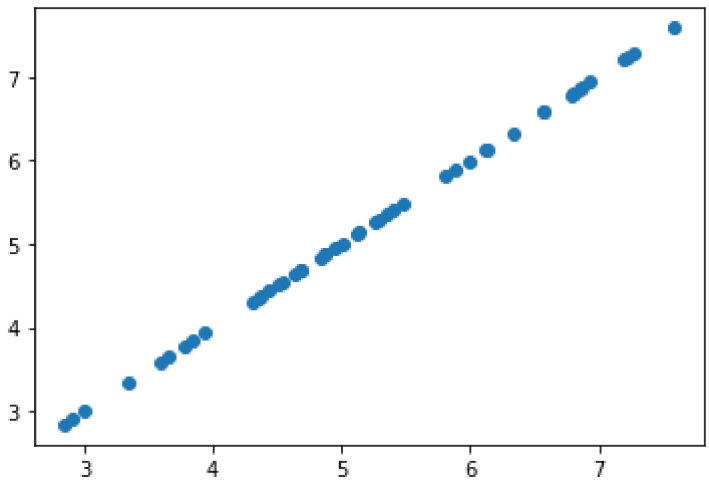
In [16]:
```
coeff =pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[16]:

	Co-efficient
Happiness Rank	-0.000006
Standard Error	-0.000721
Economy (GDP per Capita)	0.999832
Family	0.999664
Health (Life Expectancy)	0.999628
Freedom	0.999490
Trust (Government Corruption)	0.999570
Generosity	0.999750
Dystopia Residual	0.999773

In [17]:
```
prediction =lr.predict(x_test)
py.scatter(y_test,prediction)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x1ff9b8c4910>
```



In [18]:
```
print(lr.score(x_test,y_test))
```

```
0.9999999574995005
```

In [19]:
```
print(lr.score(x_train,y_train))
```

```
0.9999999318033763
```

In [20]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [21]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[21]:
```
Ridge(alpha=10)
```

In [22]:
```
rr.score(x_test,y_test)
```

Out[22]:
```
0.9787867742499708
```

In [23]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[23]:
```
Lasso(alpha=10)
```

In [24]:
```
la.score(x_test,y_test)
```

Out[24]:
```
0.9161041929327794
```

In [25]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[25]:
```
ElasticNet()
```

In [26]:
```
print(en.coef_)
```

```
[-0.02407381 -0.          0.          0.          0.          0.
  0.          0.          0.        ]
```

In [27]:
```
print(en.intercept_)
```

```
7.29994353294417
```

In [34]:
```
print(en.predict(x_test))
```

```
[6.72217205 6.57772918 3.5685027  6.77031967 5.22959572 3.83331463
 4.26664325 4.55552899 6.84254111 4.45923374 7.0351316  3.54442889
 4.89256235 4.3870123  4.31479087 6.04810532 3.52035508 5.27774334
```

```
6.216622   3.64072414 3.49628127 4.82034092 6.43328631 6.69809824
3.66479795 5.49440765 4.62775042 5.56662908 7.01105779 4.60367661
6.98698398 3.90553607 6.74624586 5.87958864 7.27586972 3.78516701
5.3981124  5.06107903 6.26476963 5.51848146 4.19442181 6.28884344
6.79439348 4.9888576  4.91663616 4.24256943 5.44626002 6.60180299]
```

In [35]:
```
print(en.score(x_test,y_test))
```

```
0.9767628327244146
```

Evaluation metrics

In [36]:
```
from sklearn import metrics
```

In [37]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.00022309987126977368
```

In [38]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 6.824130630331006e-08
```

In [39]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

```
Root Mean Squared Error: 0.00026123037017795246
```

In [ ]:
```
```