D2

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

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\2_2015.csv")
df
```

Out[2]:

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

158 rows × 12 columns

In [3]:
```
df.head(10)
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freec
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911	0.64
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284	0.61
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087	0.65
8	New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	0.90837	0.63
9	Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	0.93156	0.65

In [4]:
```
df.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]:
```
df.describe()
```

Out[5]:

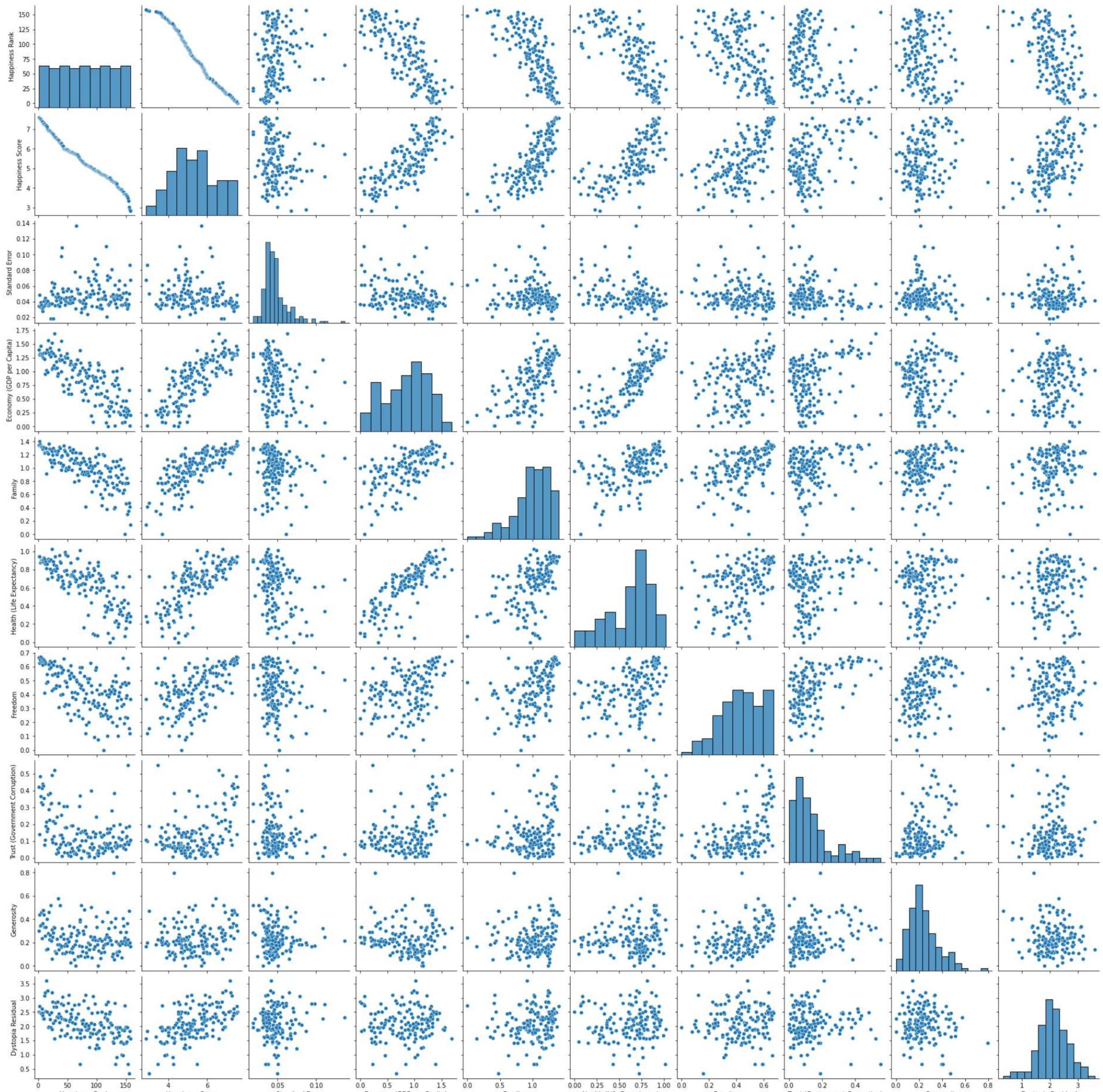
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Go C
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	1
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	
std	45.754363	1.145010	0.017146	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 [6]:
```
df.columns
```

Out[6]:
```
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 [7]:
```
sns.pairplot(df)
```

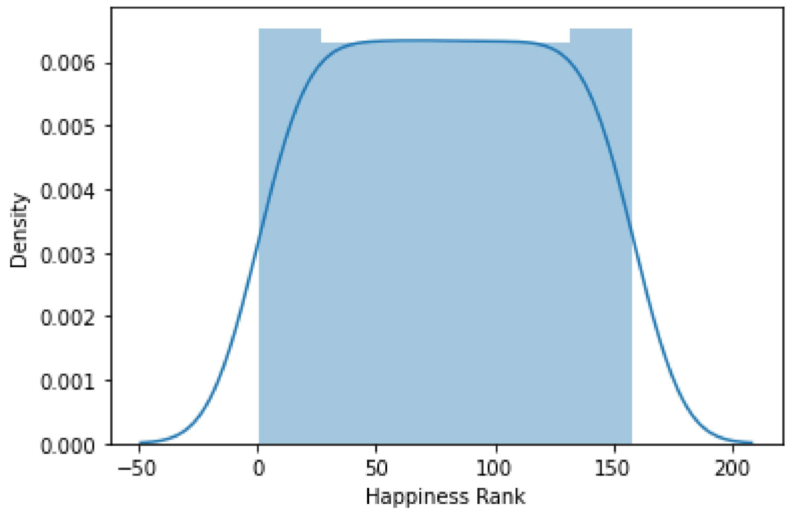
Out[7]: <seaborn.axisgrid.PairGrid at 0x26cbaea8bb0>



In [8]:
```
sns.distplot(df["Happiness Rank"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

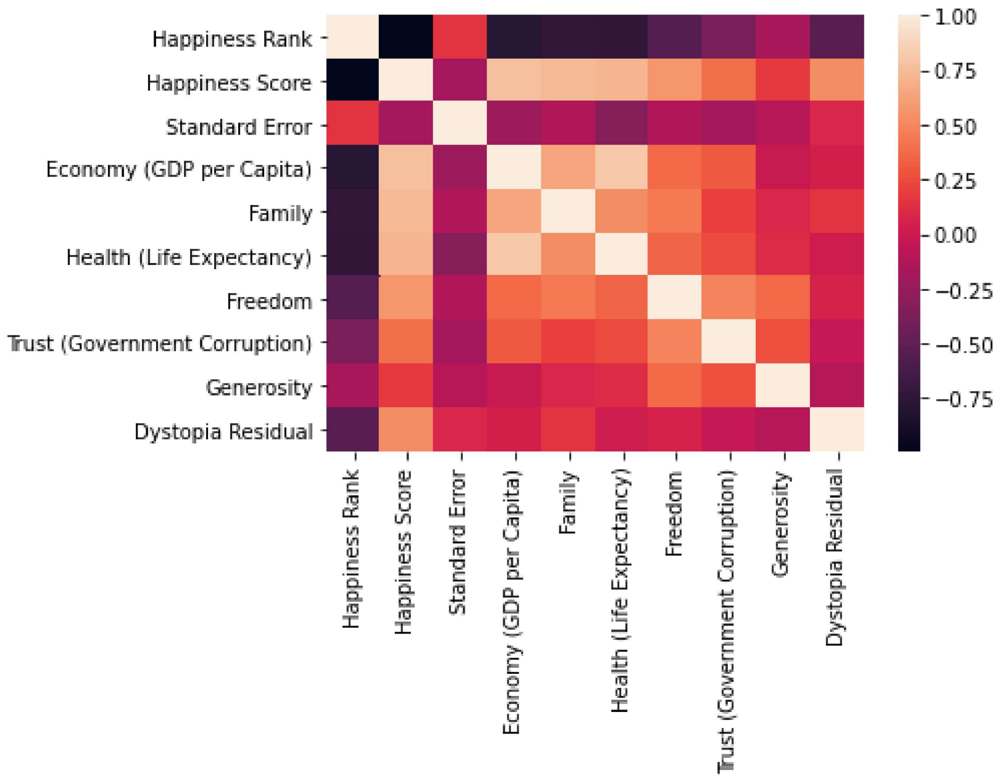
Out[8]:
```
<AxesSubplot:xlabel='Happiness Rank', ylabel='Density'>
```



In [9]:
```
df1=df[['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
```

In [10]:
```
sns.heatmap(df1.corr())
```

Out[10]: <AxesSubplot:>



In [11]:
```
x=df1[['Happiness Rank',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y=df1['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
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[13]: LinearRegression()

In [14]:
```
print(lr.intercept_)
```

```
0.002297688972016765
```

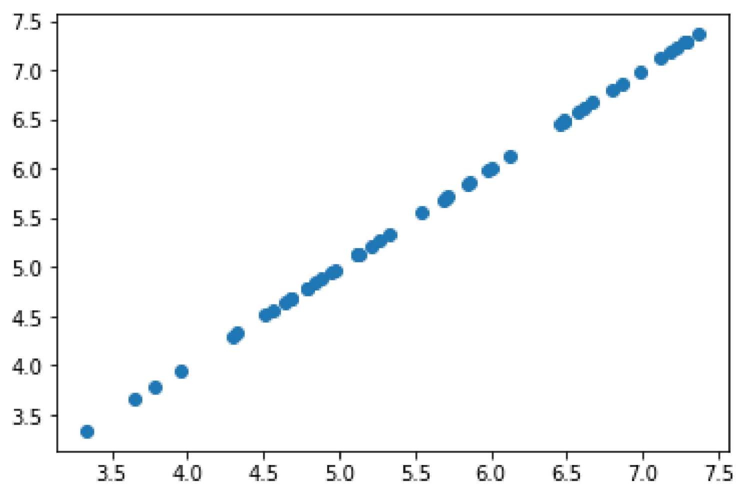
In [15]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[15]:

	Co-efficient
Happiness Rank	-0.000007
Standard Error	-0.000861
Economy (GDP per Capita)	0.999932
Family	0.999737
Health (Life Expectancy)	0.999249
Freedom	0.999453
Trust (Government Corruption)	0.999751
Generosity	0.999684
Dystopia Residual	0.999717

In [16]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]: <matplotlib.collections.PathCollection at 0x26cc1911d00>



In [17]:
```
print(lr.score(x_test,y_test))
```

```
0.9999999294713815
```

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

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

Out[19]: Ridge(alpha=10)

In [20]:
```
rr.score(x_test,y_test)
```

Out[20]: 0.9883048662525006

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

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

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

Out[22]:
```
0.9447721169215195
```

In [23]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[23]:
```
ElasticNet()
```

In [24]:
```
print(en.coef_)
```

```
[-0.02453382 -0.          0.          0.         -0.          0.
  0.          0.          0.        ]
```

In [25]:
```
print(en.intercept_)
```

```
7.321162934226761
```

In [26]:
```
print(en.predict(x_train))
```

```
[3.64109035 5.38299137 6.19260734 7.24756148 6.87955422 5.57926191
 4.67151067 5.77553245 3.59202272 5.33392374 5.43205901 5.94726917
 4.35257105 6.83048659 3.49388745 5.72646482 3.5674889  4.10723288
 7.17396003 3.78829325 4.03363143 3.91096234 7.14942621 3.86189471
 6.43794551 5.1376532  4.4261725  6.36434406 4.86778121 5.65286336
 3.76375944 3.83736089 5.62832955 3.46935363 5.30938992 6.68328369
 6.04540444 4.30350342 6.5606146  4.76964594 4.9904503  3.69015799
 5.89820154 6.21714116 6.33981025 3.98456379 7.22302767 4.47524014
 4.22990196 3.7146918  4.40163869 6.58514842 3.81282707 5.53019428
 4.05816524 3.54295508 4.25443578 4.37710487 3.44481981 4.69604449
 4.57337541 3.66562417 5.03951793 5.08858557 5.26032229 3.93549616
 6.29074261 7.29662912 5.06405175 6.38887788 7.2720953  4.2789696
 6.11900589 5.16218702 5.92273535 6.31527643 6.06993826 4.94138266
 6.14353971 4.81871358 5.8491339  4.20536815 4.52430777 5.80006627
 5.50566046 5.87366772 6.7078175  7.19849385 4.91684885 4.8432474
 7.00222331 6.78141896 3.96002998 5.55472809 4.49977395 6.90408804
 5.82460008 6.4134117  6.85502041 5.48112664 6.02087062 4.00909761
 4.64697686 6.75688514 4.08269906 6.09447207 6.46247933 5.30938992
 4.15630051 5.11311938]
```

In [27]:
```
print(en.score(x_train,y_train))
```

```
0.9822620220969362
```

In [28]:
```
from sklearn import metrics
```

In [29]:
```
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolytre Error: 0.0002543941045462239
```

In [30]:
```
print("Mean Square Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Square Error: 8.442572044849555e-08
```

In [31]:
```
print("Root Mean Square Error:",np.sqrt(metrics.mean_absolute_error(y_test,pred
```

```
Root Mean Square Error: 0.015949736817459526
```

In [ ]: