Problem Statement

Linear Regression

Import Libraries

In [1]:

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

In [2]:

```
a=pd.read_csv("2015.csv")
a
```

Out[2]:

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

To display top 10 rows

Loading [MathJax]/extensions/Safe.js

In [3]:

```
c=a.head(15)
c
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)
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
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087
8	New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	0.90837
9	Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	0.93156
10	Israel	Middle East and Northern Africa	11	7.278	0.03470	1.22857	1.22393	0.91387
11	Costa Rica	Latin America and Caribbean	12	7.226	0.04454	0.95578	1.23788	0.86027
12	Austria	Western Europe	13	7.200	0.03751	1.33723	1.29704	0.89042
13	Mexico	Latin America and Caribbean	14	7.187	0.04176	1.02054	0.91451	0.81444
14	United States	North America	15	7.119	0.03839	1.39451	1.24711	0.86179

To find Missing values

Loading [MathJax]/extensions/Safe.js

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Country                        15 non-null     object 
 1   Region                         15 non-null     object 
 2   Happiness Rank                 15 non-null     int64  
 3   Happiness Score                15 non-null     float64
 4   Standard Error                 15 non-null     float64
 5   Economy (GDP per Capita)       15 non-null     float64
 6   Family                         15 non-null     float64
 7   Health (Life Expectancy)       15 non-null     float64
 8   Freedom                        15 non-null     float64
 9   Trust (Government Corruption)  15 non-null     float64
 10  Generosity                     15 non-null     float64
 11  Dystopia Residual              15 non-null     float64
dtypes: float64(9), int64(1), object(2)
memory usage: 1.5+ KB
```

To display summary of statistics

In [5]:

```
a.describe()
```

Out[5]:

	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

To display column heading

Loading [MathJax]/extensions/Safe.js

In [6]:

```
a.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')
```

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

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

Loading [MathJax]/extensions/Safe.js

In [8]:

```
s.columns
```

Out[8]:

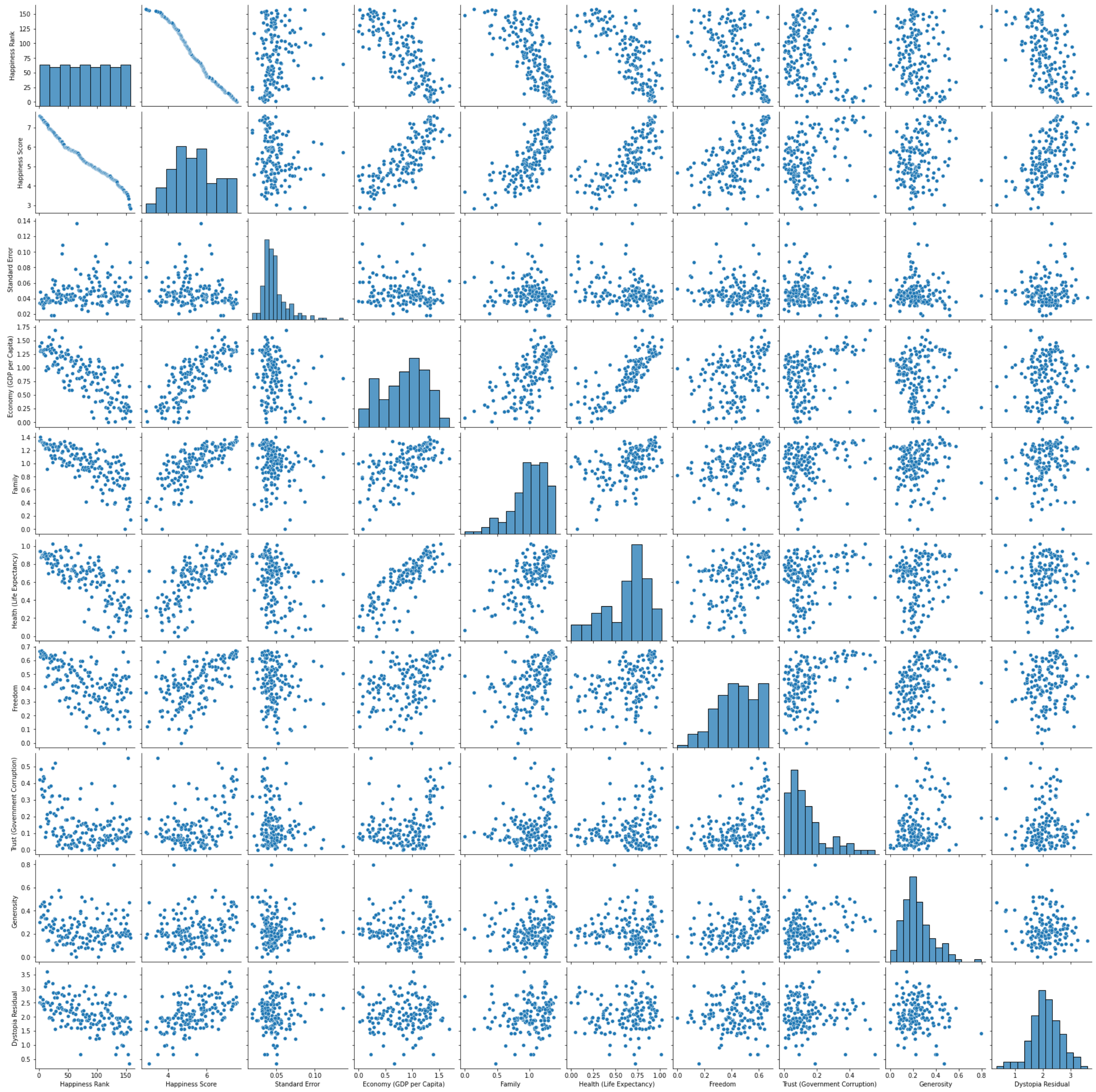
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [9]:

```
sns.pairplot(a)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x11f21df7820>
```



Distribution Plot

Loading [MathJax]/extensions/Safe.js

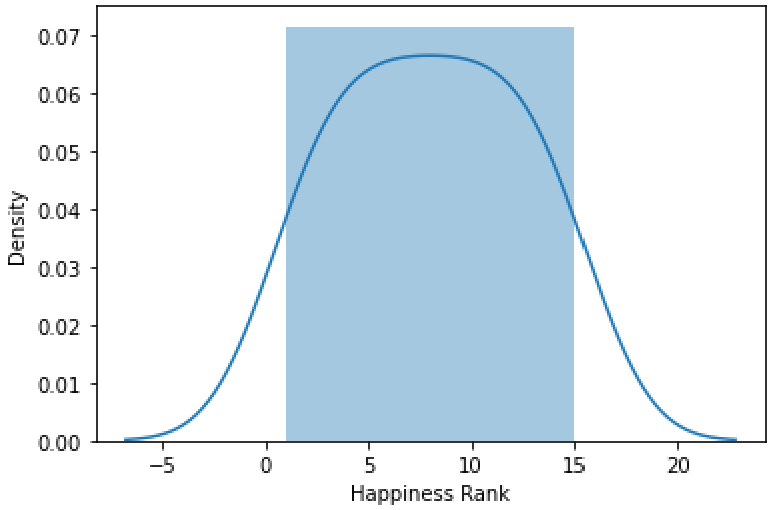
In [10]:

```
sns.distplot(c['Happiness Rank'])
```

```
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 adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]:

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



Correlation

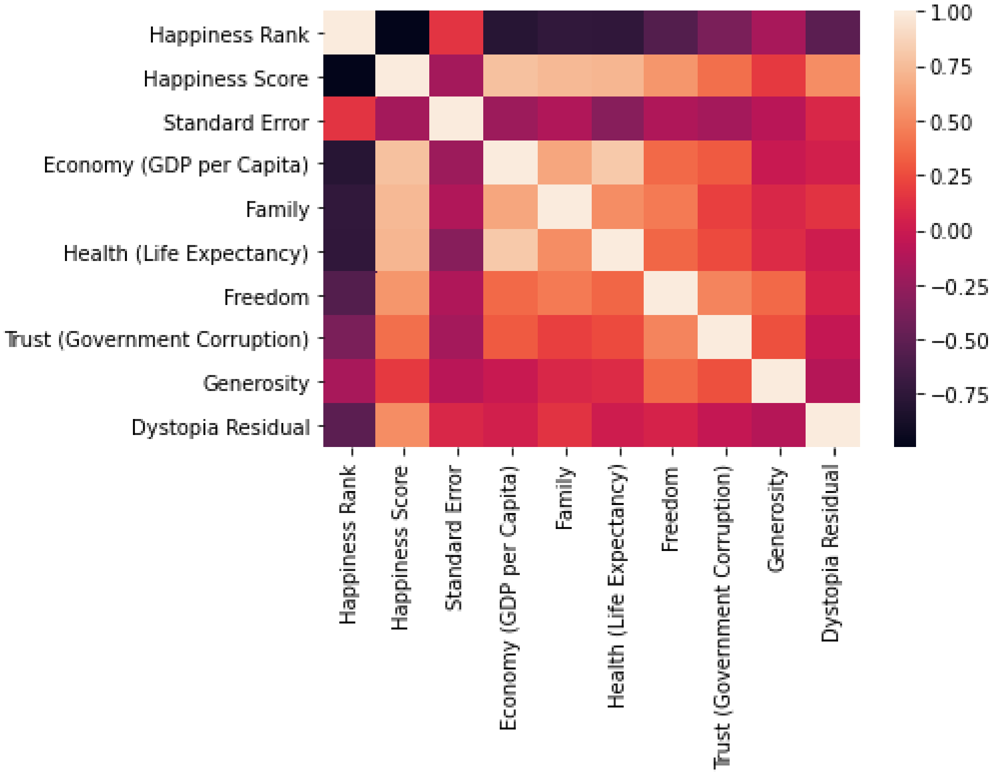
Loading [MathJax]/extensions/Safe.js

In [11]:

```
b=a[['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
sns.heatmap(b.corr())
```

Out[11]:

```
<AxesSubplot:>
```



Train the model - Model Building

In [12]:

```
g=c[['Happiness Rank']]
h=c['Happiness Rank']
```

To split dataset into training end test

In [13]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

Loading [MathJax]/extensions/Safe.js

In [14]:

```
from sklearn.linear_model import LinearRegression
```

In [15]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

```
print(lr.intercept_)
```

```
-4.440892098500626e-15
```

Coeffecient

In [17]:

```
coeff=pd.DataFrame(lr.coef_,g.columns,columns=['Co-effecient'])
coeff
```

Out[17]:

	Co-effecient
Happiness Rank	1.0

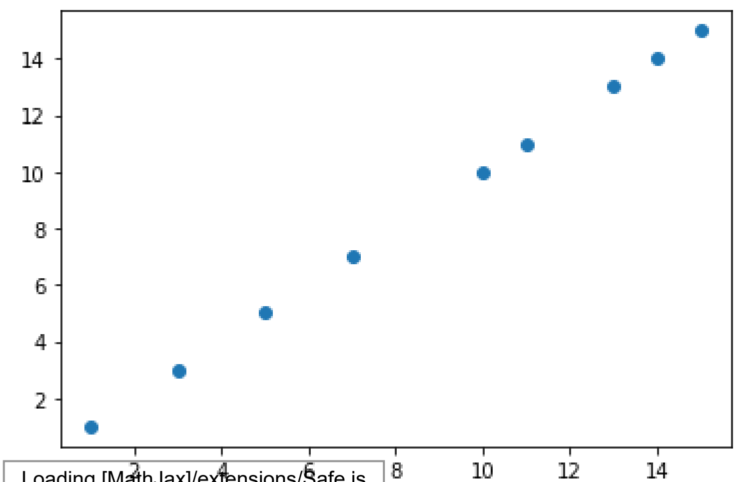
Best Fit line

In [18]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x11f288353a0>
```



Loading [MathJax]/extensions/Safe.js

To find score

In [19]:

```
print(lr.score(g_test,h_test))
```

1.0

Import Lasso and ridge

In [20]:

```
from sklearn.linear_model import Ridge,Lasso
```

Ridge

In [21]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[21]:

Ridge(alpha=5)

In [22]:

```
ri.score(g_test,h_test)
```

Out[22]:

0.9940081041950968

In [23]:

```
ri.score(g_train,h_train)
```

Out[23]:

0.9948735767055326

Lasso

In [24]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[24]:

Lasso(alpha=6)

Loading [MathJax]/extensions/Safe.js

In [25]:

```
l.score(g_test,h_test)
```

Out[25]:

```
0.6396221509642677
```

In [26]:

```
ri.score(g_train,h_train)
```

Out[26]:

```
0.9948735767055326
```

ElasticNet

In [30]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[30]:

```
ElasticNet()
```

Coeffecient,intercept

In [31]:

```
print(e.coef_)
```

```
[0.91154791]
```

In [32]:

```
print(e.intercept_)
```

```
0.604422604422604
```

Prediction

In [33]:

```
c=e.predict(g_test)
```

To calculate Score

Loading [MathJax]/extensions/Safe.js

In [35]:

```
print(e.score(g_test,h_test))
```

```
0.9908553734949039
```

Evaluation

In [36]:

```
from sklearn import metrics
```

In [38]:

```
print("Mean Absolute Error",metrics.mean_absolute_error(h_test,c))
```

```
Mean Absolute Error 0.39803439803439766
```

In [40]:

```
print("Mean Squared Error",metrics.mean_squared_error(h_test,c))
```

```
Mean Squared Error 0.204794475064745
```

In [41]:

```
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(h_test,c)))
```

```
Root Mean Squared Error 0.45254223566949525
```

In [ ]:

```

```