LinearRegression

In [1]:

```
import numpy as np
import pandas as pd
```

data collection

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pp
import seaborn as sb
```

In [3]:

```
df = pd.read_csv(r"C:\Users\user\Desktop\2015.csv")
df
```

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
...	...	...	...	...	...	...	...	...
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

first 10 rows

In [4]:

```
df.head(10)
```

Out[4]:

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

data cleaning

In [5]:

```
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 [6]:

```
df.describe()
```

Out[6]:

	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

In [7]:

```
df.columns
```

Out[7]:

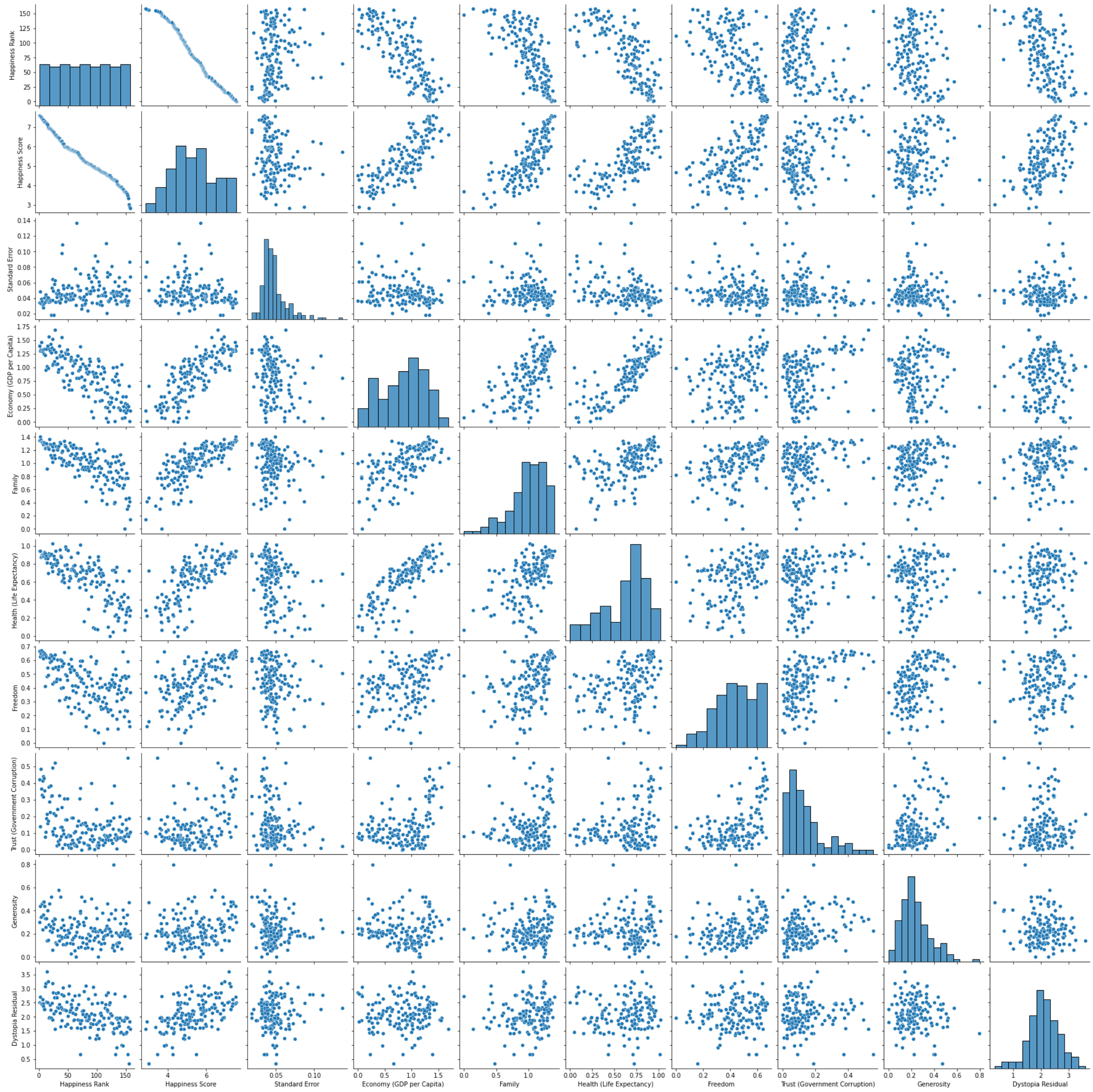
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [8]:

```
sb.pairplot(df)
```

Out[8]:

<seaborn.axisgrid.PairGrid at 0x26672a2ab80>



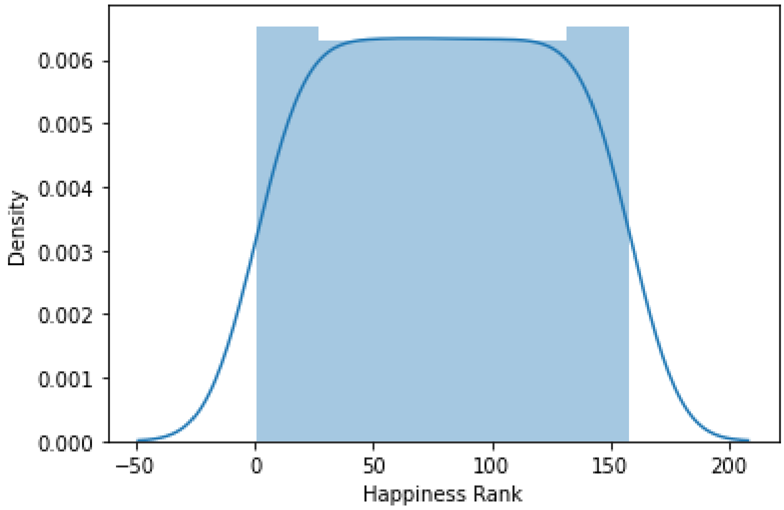
In [10]:

```
sb.distplot(df["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'>
```



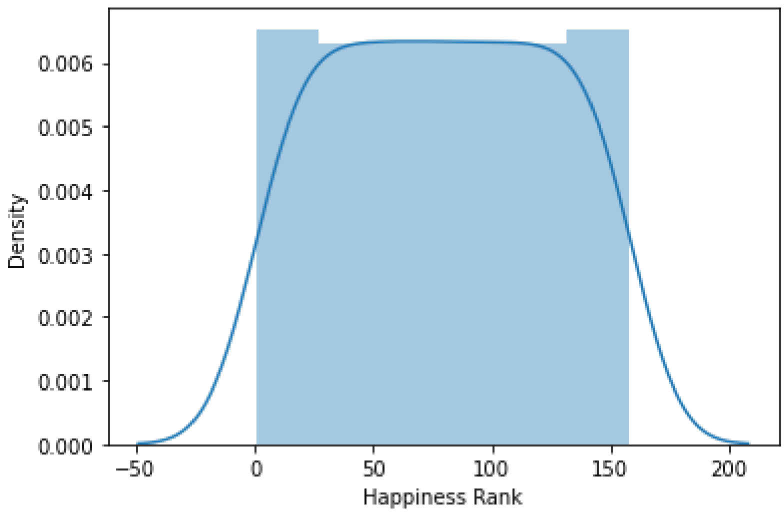
In [11]:

```
sb.distplot(df["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[11]:

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



In [14]:

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

Out[14]:

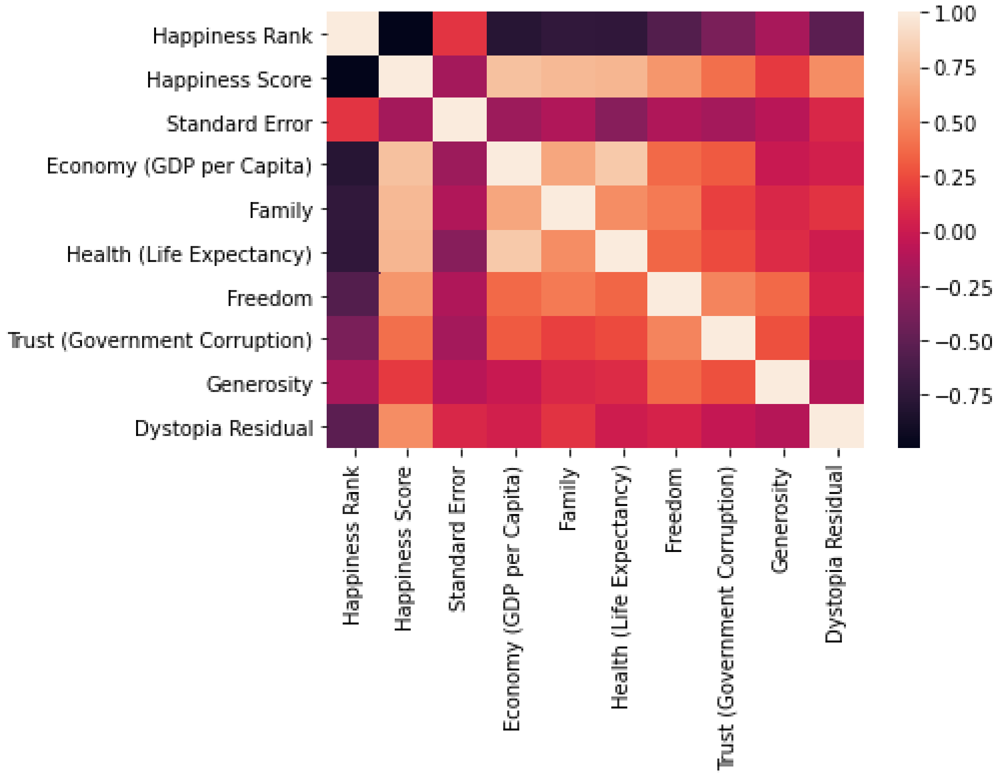
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity
0	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678
1	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630
2	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139
3	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699
4	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811
...	...	...	...	...	...	...	...	...	...
153	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628
154	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260

In [15]:

```
sb.heatmap(df1.corr())
```

Out[15]:

<AxesSubplot:>



model building

In [16]:

```
x = df1[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y = df1['Happiness Rank']
```

In [17]:

```
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)
```

linear regression

In [18]:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train,y_train)
```

Out[18]:

```
LinearRegression()
```

In [19]:

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

```
-1.2789769243681803e-13
```

In [20]:

```
coef = pd.DataFrame(lr.coef_,x.columns,columns=['Co_efficient'])
coef
```

Out[20]:

	Co_efficient
Happiness Rank	1.000000e+00
Happiness Score	1.542123e-11
Standard Error	2.235586e-13
Economy (GDP per Capita)	-1.540617e-11
Family	-1.541921e-11
Health (Life Expectancy)	-1.541812e-11
Freedom	-1.537021e-11
Trust (Government Corruption)	-1.543863e-11
Generosity	-1.541092e-11
Dystopia Residual	-1.541620e-11

In [21]:

```
print(lr.score(x_test,y_test))
```

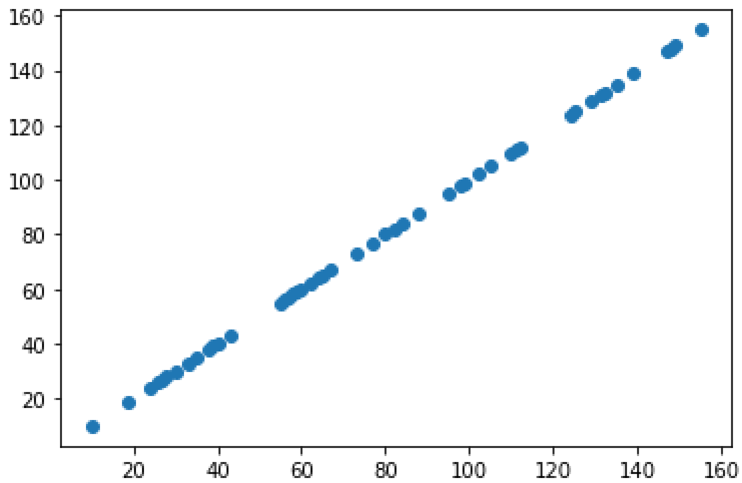
```
1.0
```

In [22]:

```
prediction = lr.predict(x_test)
pp.scatter(y_test,prediction)
```

Out[22]:

```
<matplotlib.collections.PathCollection at 0x266798fdd60>
```



lasso and ridge regression

In [23]:

```
lr.score(x_test,y_test)
```

Out[23]:

```
1.0
```

In [24]:

```
lr.score(x_train,y_train)
```

Out[24]:

```
1.0
```

In [25]:

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

In [26]:

```
r = Ridge(alpha=10)
r.fit(x_train,y_train)
r.score(x_test,y_test)
r.score(x_train,y_train)
```

Out[26]:

```
0.9999999929301249
```

In [27]:

```
l = Lasso(alpha=10)
l.fit(x_train,y_train)
l.score(x_test,y_test)
l.score(x_train,y_train)
```

Out[27]:

```
0.9999804204808944
```

elasticnet

In [28]:

```
from sklearn.linear_model import ElasticNet
e = ElasticNet()
e.fit(x_train,y_train)
```

Out[28]:

```
ElasticNet()
```

In [29]:

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

```
[ 0.99955761 -0.          0.         -0.          -0.          -0.
 -0.          -0.         -0.         -0.         ]
```

In [30]:

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

```
0.03512975817922381
```

In [31]:

```
predictions = e.predict(x_test)
predictions
```

Out[31]:

```
array([ 59.00902877,  28.02274285,  30.02185807, 128.9780615 ,
       146.97009848,  40.01743417,  26.02362763, 110.98602451,
       104.98867885,  57.00991355,  39.01787656,  27.02318524,
        64.00681682,  94.99310274,  81.99885381,  98.99133319,
        60.00858638,  55.01079833,  33.0205309 , 109.9864669 ,
       138.9736376 ,  67.00548965,  56.01035594,  38.01831895,
       131.97673433,  10.03070586,  87.99619947,  24.02451241,
        58.00947116,  77.00106576, 111.98558212, 101.99000602,
       130.97717672, 134.97540716,  62.0077016 ,  65.00637443,
        79.99973859,  83.99796903, 148.9692137 ,  97.99177557,
        43.016107  ,  35.01964612, 123.98027344,  19.02672436,
       124.97983105,  73.00283532, 154.96655937, 147.96965609])
```

In [32]:

```
print(e.score(x_test,y_test))
```

```
0.9999998042823239
```

In [33]:

```
from sklearn import metrics
```

mean absolute error

In [34]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,predictions))
```

```
Mean Absolute Error: 0.015610991566379906
```

mean squared error

In [35]:

```
print("Mean Squared Error:", metrics.mean_squared_error(y_test,predictions))
```

```
Mean Squared Error: 0.0003265072872140236
```

root mean squared error

In [36]:

```
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

```
Root Mean Squared Error 0.018069512644618382
```

In [ ]:

```

```