In [76]:
```
# import libaries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [272]:
```
x=pd.read_csv(r"C:\Users\user\Downloads\2015 - 2015.csv")
```

Out[272]:

	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 [273]:
```
x=x.head(10)
```

Out[273]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Free
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.6
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.6
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.6
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.6
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.6
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911	0.6
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284	0.6
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087	0.6
8	New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	0.90837	0.6
9	Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	0.93156	0.6

In [274]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Country                        10 non-null     object
 1   Region                         10 non-null     object
 2   Happiness Rank                 10 non-null     int64
 3   Happiness Score                10 non-null     float64
 4   Standard Error                 10 non-null     float64
 5   Economy (GDP per Capita)       10 non-null     float64
 6   Family                         10 non-null     float64
 7   Health (Life Expectancy)       10 non-null     float64
 8   Freedom                        10 non-null     float64
 9   Trust (Government Corruption)  10 non-null     float64
 10  Generosity                     10 non-null     float64
 11  Dystopia Residual              10 non-null     float64
dtypes: float64(9), int64(1), object(2)
memory usage: 1.1+ KB
```

In [275]:

Out[275]:
```
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 [276]:
```
d=x[[ 'Happiness Rank', 'Happiness Score','Standard Error']]
```

Out[276]:

	Happiness Rank	Happiness Score	Standard Error
0	1	7.587	0.03411
1	2	7.561	0.04884
2	3	7.527	0.03328
3	4	7.522	0.03880
4	5	7.427	0.03553
5	6	7.406	0.03140
6	7	7.378	0.02799
7	8	7.364	0.03157
8	9	7.286	0.03371
9	10	7.284	0.04083

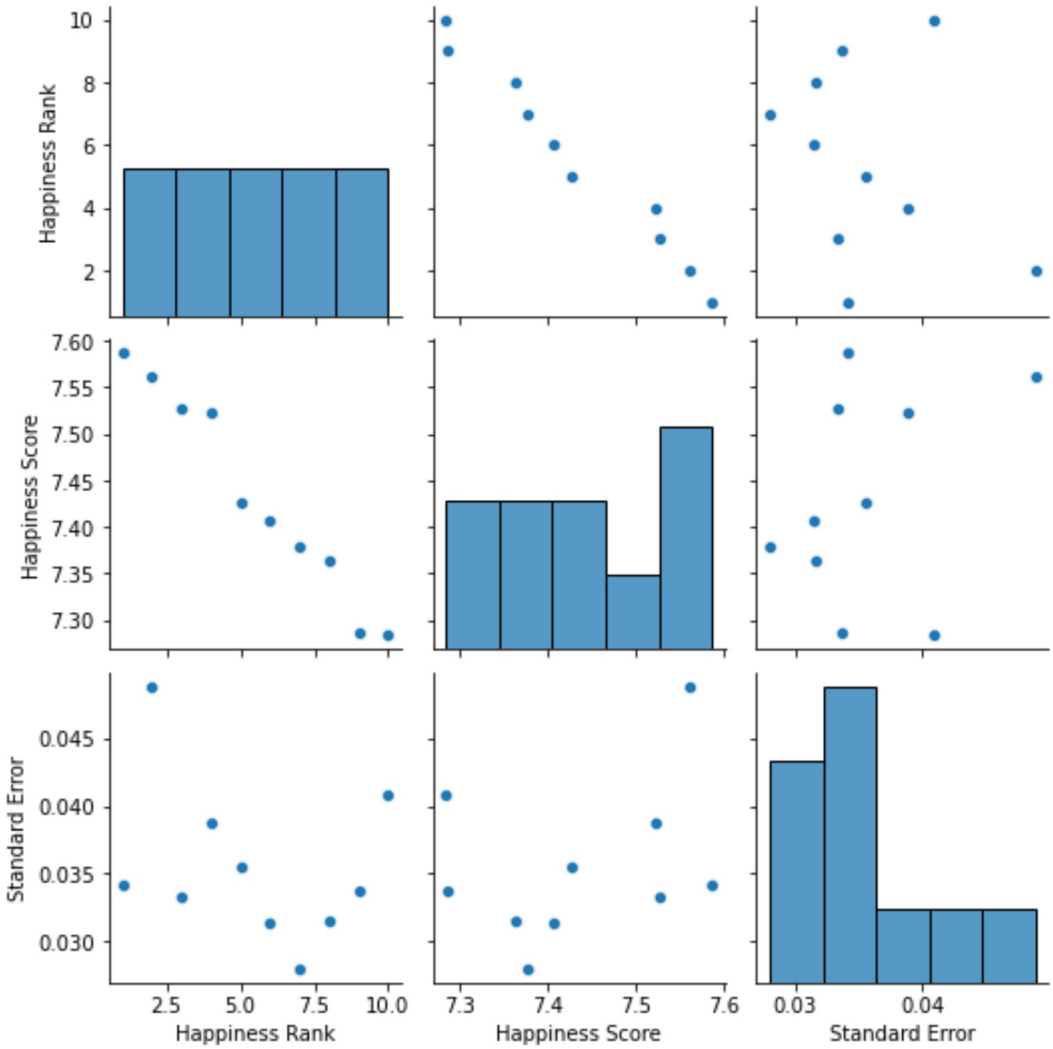
In [277]:

Out[277]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Governi Corrup
count	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00
mean	5.50000	7.434200	0.035606	1.334476	1.328228	0.908750	0.645429	0.36
std	3.02765	0.110153	0.005924	0.057380	0.035577	0.024692	0.017048	0.09
min	1.00000	7.284000	0.027990	1.250180	1.280170	0.874640	0.615760	0.14
25%	3.25000	7.367500	0.031997	1.308110	1.311487	0.890042	0.634572	0.33
50%	5.50000	7.416500	0.033910	1.327865	1.321140	0.907000	0.645535	0.38
75%	7.75000	7.525750	0.037983	1.333112	1.344870	0.926388	0.657660	0.42
max	10.00000	7.587000	0.048840	1.459000	1.402230	0.947840	0.669730	0.48

In [278]:

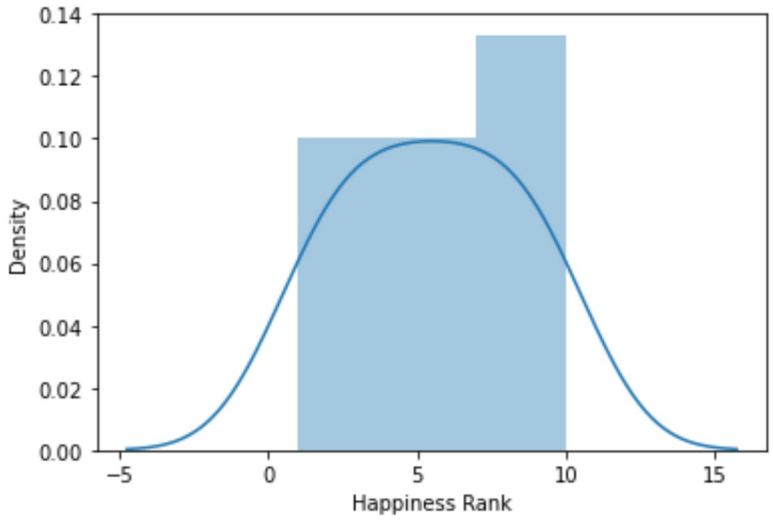
Out[278]: <seaborn.axisgrid.PairGrid at 0x190cee649d0>



In [279]:

```
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)
```

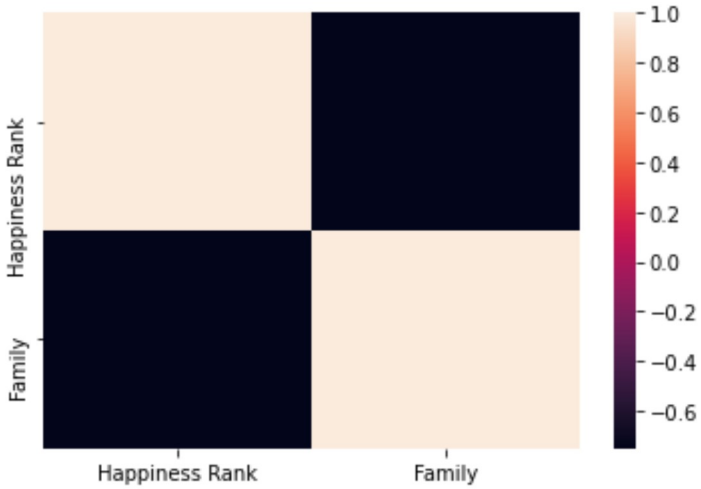
Out[279]: <AxesSubplot:xlabel='Happiness Rank', ylabel='Density'>



In [280]:

In [281]:

Out[281]: <AxesSubplot:>



In [283]:

```
x=x1[['Family','Happiness Rank']]
```

In [284]:
```
# to split my dataset into traning and test date

from sklearn.model_selection import train_test_split
```

In [285]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
```

Out[285]:
```
LinearRegression()
```

In [286]:

```
-6.661338147750939e-16
```

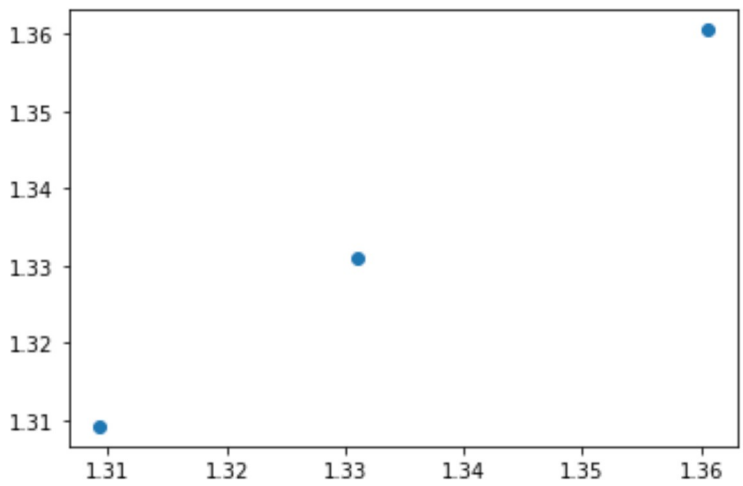
In [287]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
```

Out[287]:

	Co-efficient
Family	1.000000e+00
Happiness Rank	-1.239678e-18

In [288]:
```
prediction=lr.predict(x_test)
```

Out[288]:
```
<matplotlib.collections.PathCollection at 0x190cfc0de80>
```



In [289]:

Out[289]:
```
1.0
```

In [290]:

Out[290]:
```
1.0
```

In [291]:

In [292]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[292]: 0.4138105904194439

In [293]:
```
la=Lasso(alpha=10)
```

Out[293]: Lasso(alpha=10)

In [294]:

Out[294]: -0.13230193846194815

In [295]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
```

Out[295]: ElasticNet()

In [296]:

Out[296]: array([0., -0.])

In [297]:

Out[297]: array([1.32593143, 1.32593143, 1.32593143])

In [298]:

Out[298]: 1.3259314285714285

In [299]:

Out[299]: -0.13230193846194815

In [300]:

In [301]:

Mean Absolute Error 0.0

In [302]:

Mean Squared Error 0.0

In [303]:

Root Mean Squared Error 0.0

In [ ]: