In [38]:

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

In [39]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\2015 - 2015.csv")
a
```

Out[39]:

	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

In [40]:

```
a=a.head(10)
a
```

Out[40]:

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

In [41]:

```
# to find
a.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 [42]:

```
# to display summary of statastic
a.describe()
```

Out[42]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Gov Cor
count	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10
mean	5.50000	7.434200	0.035606	1.334476	1.328228	0.908750	0.645429	0
std	3.02765	0.110153	0.005924	0.057380	0.035577	0.024692	0.017048	0
min	1.00000	7.284000	0.027990	1.250180	1.280170	0.874640	0.615760	0
25%	3.25000	7.367500	0.031997	1.308110	1.311487	0.890042	0.634572	0
50%	5.50000	7.416500	0.033910	1.327865	1.321140	0.907000	0.645535	0
75%	7.75000	7.525750	0.037983	1.333112	1.344870	0.926388	0.657660	0
max	10.00000	7.587000	0.048840	1.459000	1.402230	0.947840	0.669730	0

In [43]:

```
# to display colum heading
a.columns
```

Out[43]:

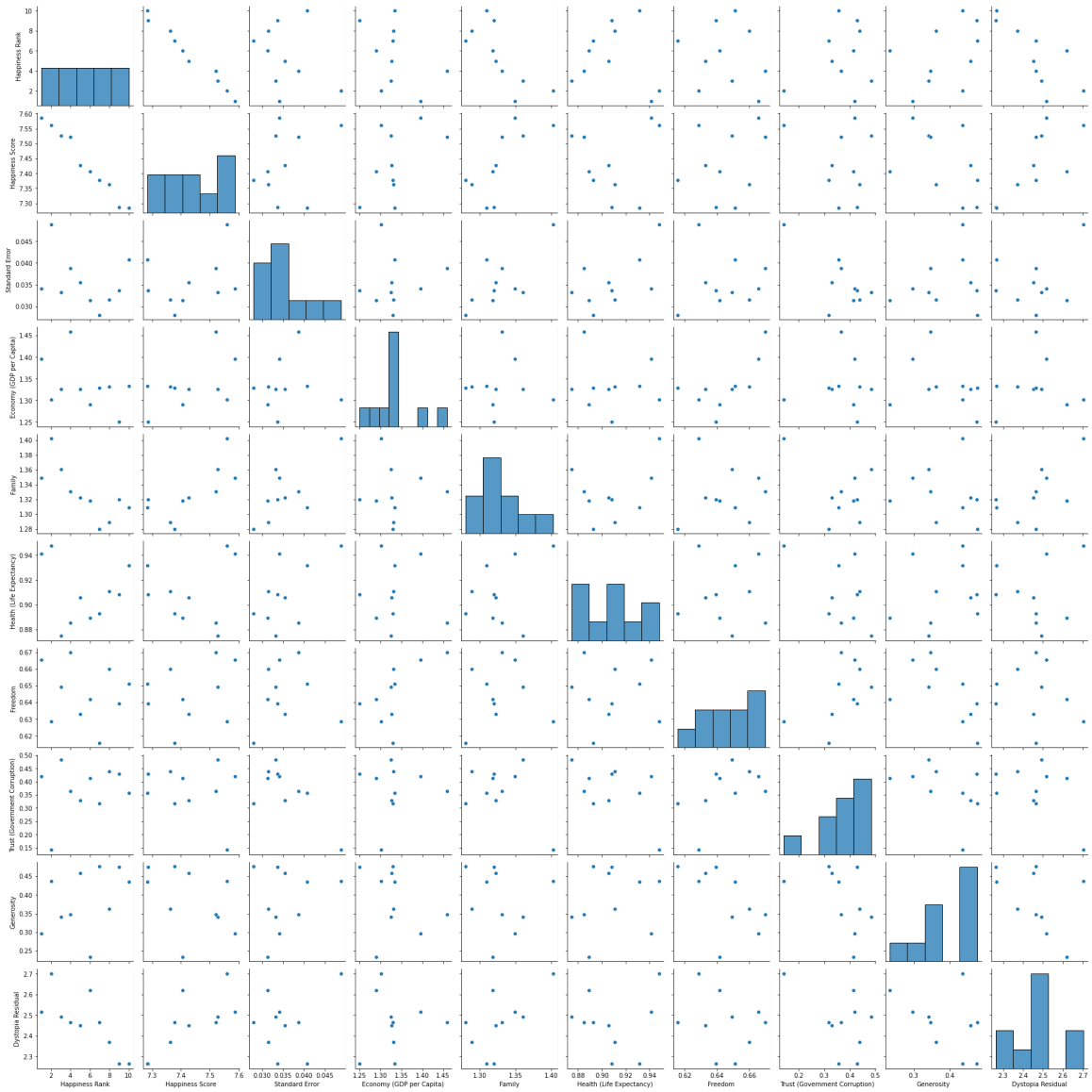
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [44]:

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

Out[44]:

```
<seaborn.axisgrid.PairGrid at 0x243e5563220>
```

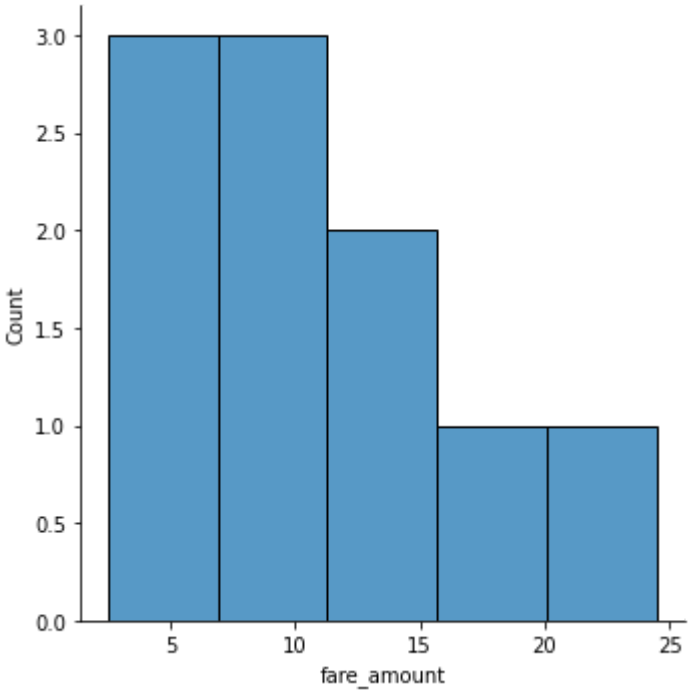


In [8]:

```
sns.displot(a["Happiness Rank"])
```

Out[8]:

```
<seaborn.axisgrid.FacetGrid at 0x243e4469490>
```



In [45]:

```
b=a[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)']]
b
```

Out[45]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)
0	1	7.587	0.03411	1.39651	1.34951	0.94143
1	2	7.561	0.04884	1.30232	1.40223	0.94784
2	3	7.527	0.03328	1.32548	1.36058	0.87464
3	4	7.522	0.03880	1.45900	1.33095	0.88521
4	5	7.427	0.03553	1.32629	1.32261	0.90563
5	6	7.406	0.03140	1.29025	1.31826	0.88911
6	7	7.378	0.02799	1.32944	1.28017	0.89284
7	8	7.364	0.03157	1.33171	1.28907	0.91087
8	9	7.286	0.03371	1.25018	1.31967	0.90837
9	10	7.284	0.04083	1.33358	1.30923	0.93156

In [46]:

```
sns.heatmap(b.corr())
```

Out[46]:

<AxesSubplot:>

In [48]:

```
x=a[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)']]
y=a['Family']
```

In [49]:

```
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 [50]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[50]:

```
LinearRegression()
```

In [51]:

```
lr.intercept_
```

Out[51]:

```
6.439293542825908e-15
```

In [52]:

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

Out[52]:

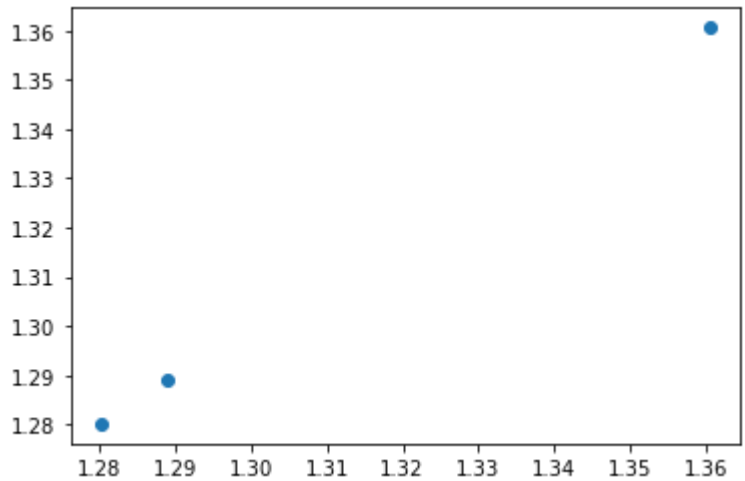
	Co-efficient
Happiness Rank	3.042081e-17
Happiness Score	-1.013378e-15
Standard Error	4.151239e-16
Economy (GDP per Capita)	1.071578e-16
Family	1.000000e+00
Health (Life Expectancy)	-1.798211e-16

In [53]:

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

Out[53]:

```
<matplotlib.collections.PathCollection at 0x243ea31fc70>
```



In [54]:

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

Out[54]:

```
1.0
```

In [55]:

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

Out[55]:

```
1.0
```

In [56]:

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

In [57]:

```
rr=Ridge(alpha=10)
rr.fit(x_test,y_test)
```

Out[57]:

```
Ridge(alpha=10)
```

In [58]:

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

Out[58]:

```
0.7586157601980559
```

In [59]:

```
la=Lasso(alpha=10)
la.fit(x_test,y_test)
```

Out[59]:

```
Lasso(alpha=10)
```

In [60]:

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

Out[60]:

```
0.0
```

In [61]:

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

Out[61]:

```
ElasticNet()
```

In [62]:

```
en.coef_
```

Out[62]:

```
array([-0.,  0.,  0.,  0.,  0.,  0.])
```

In [63]:

```
en.intercept_
```

Out[63]:

```
1.3360657142857144
```

In [64]:

```
prediction=en.predict(x_test)
prediction
```

Out[64]:

```
array([1.33606571, 1.33606571, 1.33606571])
```

In [65]:

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

Out[65]:

```
-0.5269025317552469
```

EVALUATION METRICS

In [66]:

```
from sklearn import metrics
```

In [67]:

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

```
Mean Absolute Error: 0.042468571428571446
```

In [68]:

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

```
Mean Squared Error 0.0019779594136054477
```

In [69]:

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

```
Root Mean Squared Error 0.044474255627333974
```

In [ ]: