In [201]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Just going to try training data for this code

In [202]:
```python
data = pd.read_csv("2015.csv")
data
```

Out[202]:

	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	(
...	...	...	...	...	...	...	...	...	
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 [203]: data.describe()
```

Out[203]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(G
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	

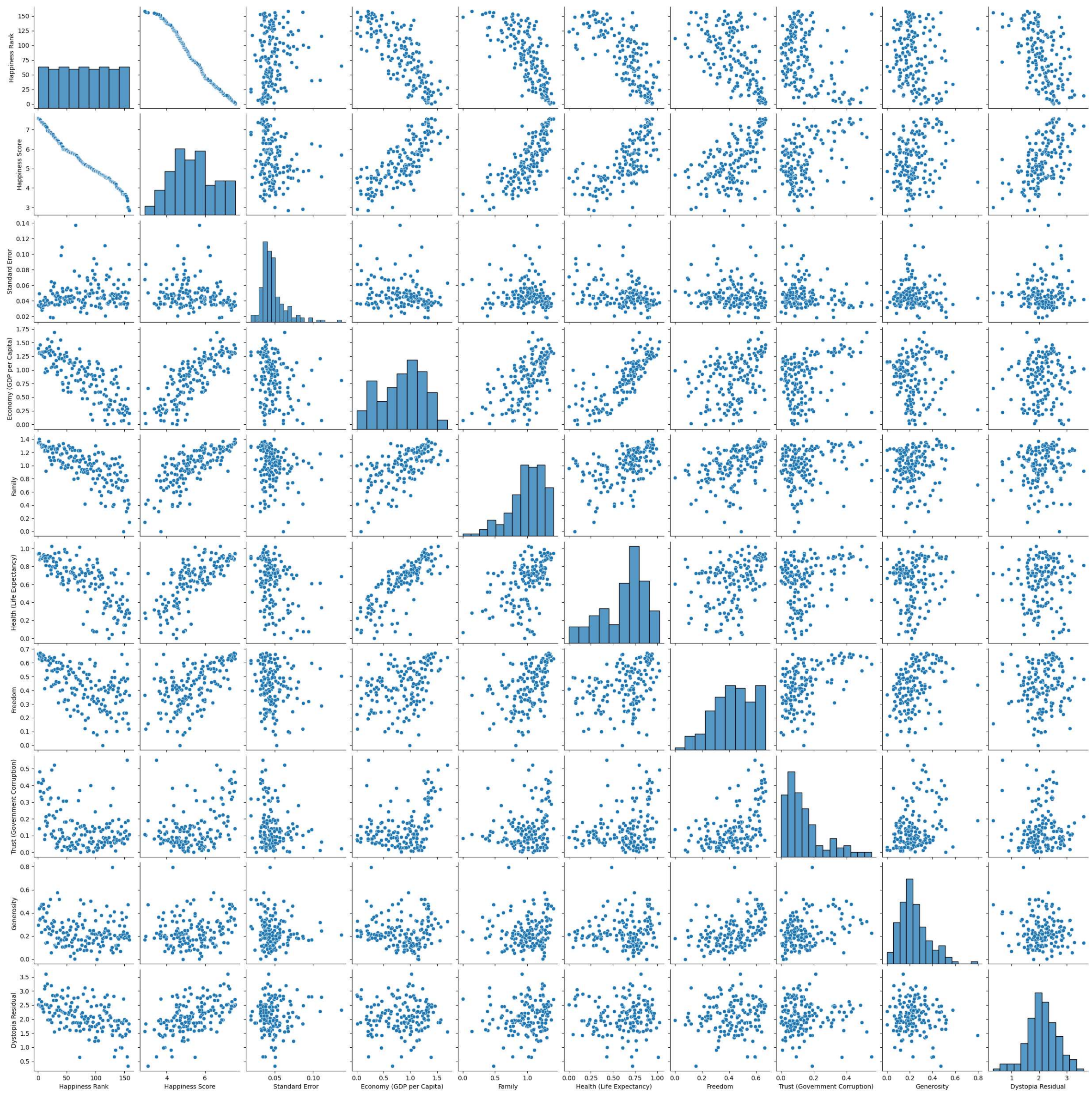
```
In [204]: data.corr()
```

Out[204]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(G
Happiness Rank	1.000000	-0.992105	0.158516	-0.785267	-0.733644	-0.735613	-0.556886	
Happiness Score	-0.992105	1.000000	-0.177254	0.780966	0.740605	0.724200	0.568211	
Standard Error	0.158516	-0.177254	1.000000	-0.217651	-0.120728	-0.310287	-0.129773	
Economy (GDP per Capita)	-0.785267	0.780966	-0.217651	1.000000	0.645299	0.816478	0.370300	
Family	-0.733644	0.740605	-0.120728	0.645299	1.000000	0.531104	0.441518	
Health (Life Expectancy)	-0.735613	0.724200	-0.310287	0.816478	0.531104	1.000000	0.360477	
Freedom	-0.556886	0.568211	-0.129773	0.370300	0.441518	0.360477	1.000000	
Trust (Government Corruption)	-0.372315	0.395199	-0.178325	0.307885	0.205605	0.248335	0.493524	
Generosity	-0.160142	0.180319	-0.088439	-0.010465	0.087513	0.108335	0.373916	
Dystopia Residual	-0.521999	0.530474	0.083981	0.040059	0.148117	0.018979	0.062783	

In [205]:
```
sns.pairplot(data)
```

Out[205]: <seaborn.axisgrid.PairGrid at 0x174cda387c0>



GDP per Capita closely correlated to Happiness Score

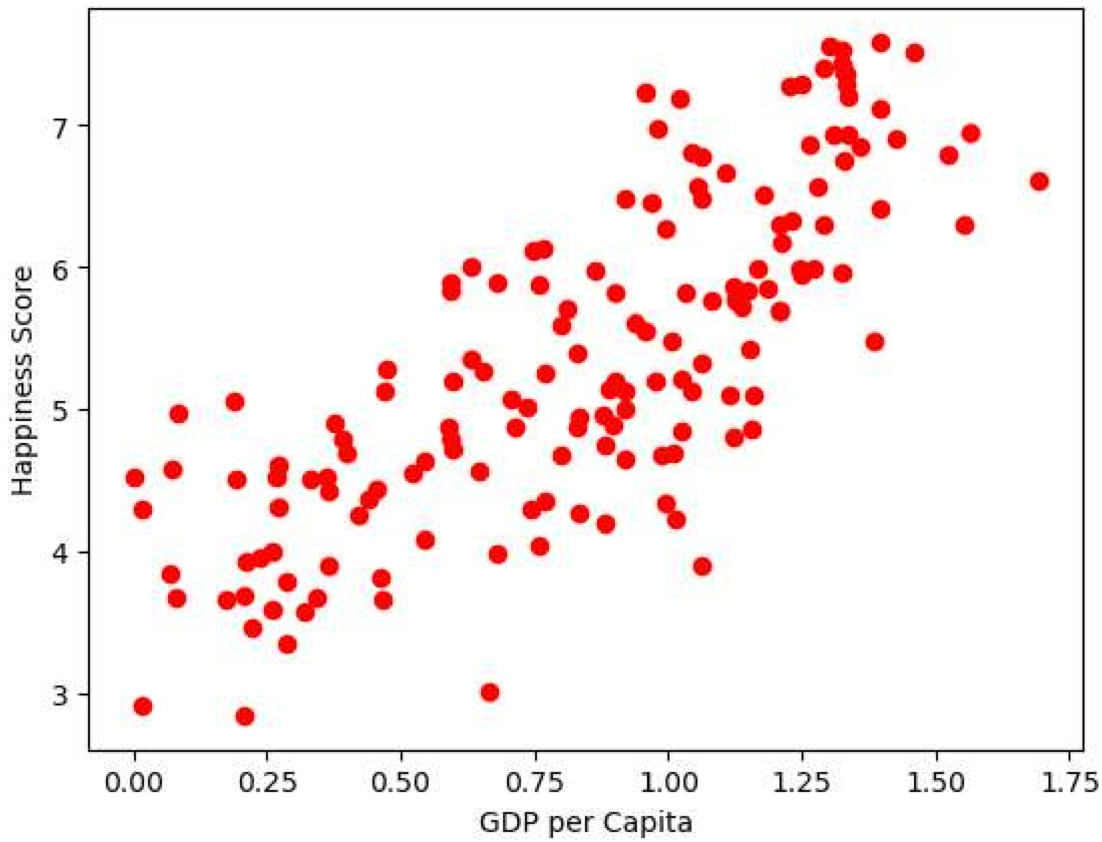
In [206]:
```
x = data['Economy (GDP per Capita)']
x
```

Out[206]:
```
0      1.39651
1      1.30232
2      1.32548
3      1.45900
4      1.32629
        ...
153    0.22208
154    0.28665
155    0.66320
156    0.01530
157    0.20868
Name: Economy (GDP per Capita), Length: 158, dtype: float64
```

In [207]:
```
y = data['Happiness Score']
y
```

Out[207]:
```
0      7.587
1      7.561
2      7.527
3      7.522
4      7.427
       ...
153    3.465
154    3.340
155    3.006
156    2.905
157    2.839
Name: Happiness Score, Length: 158, dtype: float64
```

In [208]:
```
plt.scatter(x, y, c = 'red')
plt.xlabel('GDP per Capita')
plt.ylabel('Happiness Score')
plt.show()
```



In [209]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, ra
```

In [210]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(126,)
(32,)
(126,)
(32,)
```

In [211]:
```
x_train = x_train.values.reshape(-1, 1)
```

In [212]:
```
model = LinearRegression()
model.fit(x_train,y_train)
model.score(x_train,y_train)
```

Out[212]: 0.5853418820714025

In [213]:
```
r_sq = model.score(x_train,y_train)
print(f"coefficient of determination: {r_sq}")
```

```
coefficient of determination: 0.5853418820714025
```

In [214]:
```
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
```

```
intercept: 3.505459971623436
slope: [2.18161783]
```

In [215]:
```
y_pred = model.intercept_ + model.coef_*x_train
print(f"Predicted response:\n {y_pred}")
```

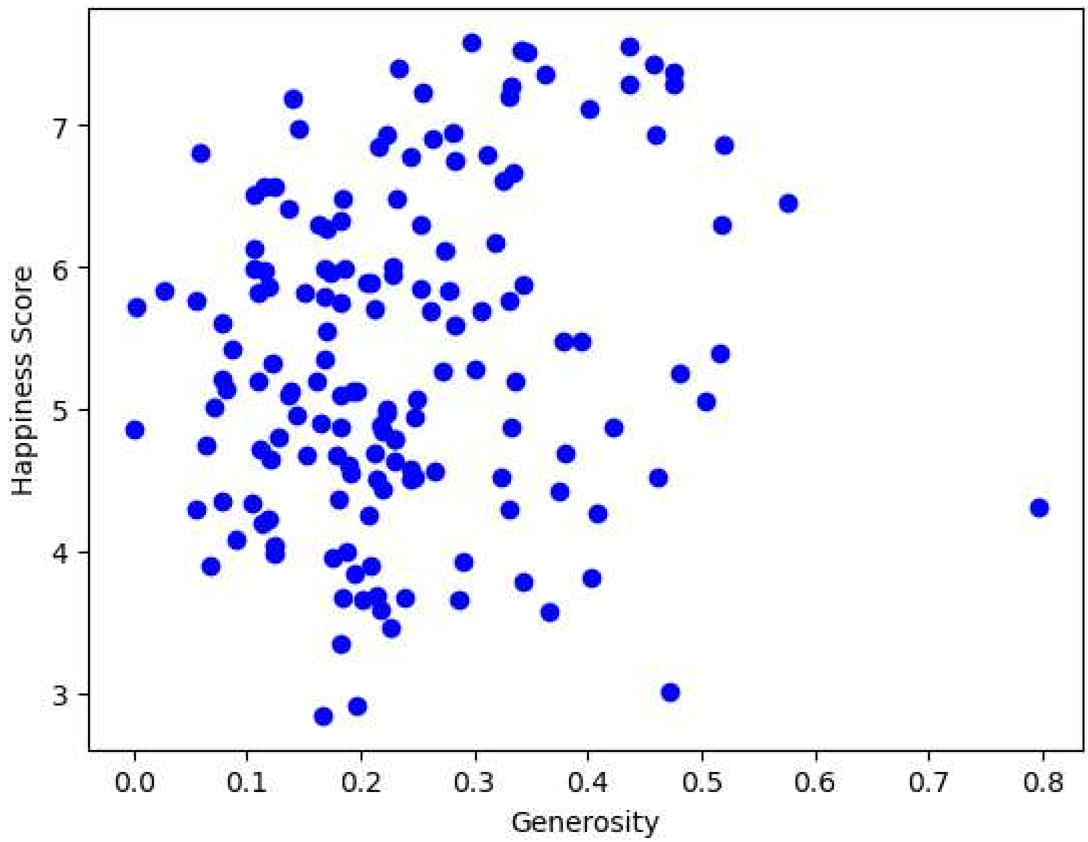
```
Predicted response:
 [[4.6976923 ]
 [6.05557487]
 [5.67300636]
 [5.78186909]
 [4.9330016 ]
 [5.95092266]
 [5.61486625]
 [4.20318499]
 [4.79970475]
 [5.15864633]
 [4.4271935 ]
 [6.42278478]
 [6.09063346]
 [4.8042207 ]
 [6.2930876 ]
 [5.59647521]
 [4.0966784 ]
 [3.54045312]
```

Let's check the independent variable Generosity

In [216]:
```
x1 = data['Generosity']
x1
```

Out[216]:
```
0      0.29678
1      0.43630
2      0.34139
3      0.34699
4      0.45811
         ...
153    0.22628
154    0.18260
155    0.47179
156    0.19727
157    0.16681
Name: Generosity, Length: 158, dtype: float64
```

In [217]:
```
plt.scatter(x1, y, c = "blue")
plt.xlabel("Generosity")
plt.ylabel("Happiness Score")
plt.show()
```



In [218]:
```
x1_train, x1_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2,
```

In [219]:
```
print(x1_train.shape)
print(x1_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(126,)
(32,)
(126,)
(32,)
```

In [220]:
```
x1_train = x1_train.values.reshape(-1,1)
```

In [221]:
```
model1 = LinearRegression()
model1.fit(x1_train,y_train)
model1.score(x1_train,y_train)
```

Out[221]:
```
0.044169945197161664
```

In [222]:
```
r1_sq = model1.score(x1_train,y_train)
print(f"coefficient of determination: {r1_sq}")
```

```
coefficient of determination: 0.044169945197161664
```

In [223]:
```
print(f"intercept: {model1.intercept_}")
print(f"slope: {model1.coef_}")
```

```
intercept: 4.912911161122698
slope: [1.88238683]
```

In [224]:
```
y1_pred = model1.intercept_ + model1.coef_*x1_train
print(f"Predicted Response:\n {y1_pred}")
```

```
Predicted Response:
 [[5.0847919 ]
 [5.23090277]
 [5.10982765]
 [5.1717017 ]
 [5.42554157]
 [5.15385668]
 [5.99773069]
 [5.60017059]
 [5.43649706]
 [5.14527299]
 [5.30102168]
 [5.53575532]
 [5.3896821 ]
 [5.34636838]
 [5.14504711]
 [5.23252162]
 [5.27031995]
 [5.53650827]
```

Multi Linear Regression taking Economy, family and Health

In [225]:
```python
x2 = data.iloc[:,5:8]
x2
```

Out[225]:

	Economy (GDP per Capita)	Family	Health (Life Expectancy)
0	1.39651	1.34951	0.94143
1	1.30232	1.40223	0.94784
2	1.32548	1.36058	0.87464
3	1.45900	1.33095	0.88521
4	1.32629	1.32261	0.90563
...	...	...	...
153	0.22208	0.77370	0.42864
154	0.28665	0.35386	0.31910
155	0.66320	0.47489	0.72193
156	0.01530	0.41587	0.22396
157	0.20868	0.13995	0.28443

158 rows × 3 columns

In [226]:
```python
y
```

Out[226]:
```
0      7.587
1      7.561
2      7.527
3      7.522
4      7.427
       ...
153    3.465
154    3.340
155    3.006
156    2.905
157    2.839
Name: Happiness Score, Length: 158, dtype: float64
```

In [227]:
```python
x2_train, x2_test, y_train, y_test = train_test_split(x2, y, test_size=0.2, r
```

In [228]:
```python
print(x2_train.shape)
print(x2_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(126, 3)
(32, 3)
(126,)
(32,)
```

In [230]:
```
modelmulti = LinearRegression()
modelmulti.fit(x2_train,y_train)
```

Out[230]:
```
LinearRegression()
```

In [231]:
```
modelmulti.score(x2_train,y_train)
```

Out[231]:
```
0.7146903461180147
```

In [232]:
```
print(f"intercept: {modelmulti.intercept_}")
print(f"slope: {modelmulti.coef_}")
```

```
intercept: 2.232075673796563
slope: [0.81089902 1.70486407 1.21518245]
```

In [234]:
```
ymulti_pred = modelmulti.intercept_ + modelmulti.coef_*x2_train
print(f"Predicted scores are: {ymulti_pred}")
```

```
Predicted scores are: [[2.67522388 3.39296876 2.71892637]
 [3.17994365 4.39723599 3.19087893]
 [3.03774439 4.11533672 2.29011279]
 [3.07820825 3.74238065 3.16642946]
 [2.76268745 3.77381835 2.42658993]
 [3.14104482 4.28157801 3.15445991]
 [3.01613393 4.38879691 3.12948791]
 [2.4914174  2.74839376 2.60070127]
 [2.71314152 4.17875766 3.13512636]
 [2.84655873 3.69894072 2.43480456]
 [2.57468051 3.74543236 2.51645267]
 [3.31643417 4.44335256 3.31409843]
 [3.19297479 4.40381676 3.29337957]
 [2.71482008 3.85762946 3.07674899]
 [3.26822622 4.38085225 3.38138308]
 [3.00929806 4.32339833 2.88688889]
 [2.45182931 3.99279109 2.638858  ]
 [2.24508249 2.93335446 2.50624514]
 [3.31540433 4.56685292 3.32006497]
```