World Happiess Score Prediction ¶

In [57]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
import pickle as pkl
from sklearn.metrics import mean_squared_error, r2_score
```

In [58]:
```
df = pd.read_csv('happiness_score_dataset.csv')
```

In [59]:
```
df.head()
```

Out[59]:

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

In [60]:
```
df.tail()
```

Out[60]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Gc C
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	

In [61]:
```
df.shape
```

Out[61]:
```
(158, 12)
```

In [62]:
```
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 [63]:
```
df.isnull().sum()
```

Out[63]:
```
Country                          0
Region                           0
Happiness Rank                   0
Happiness Score                  0
Standard Error                   0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          0
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```

In [64]:
```
df.describe()
```

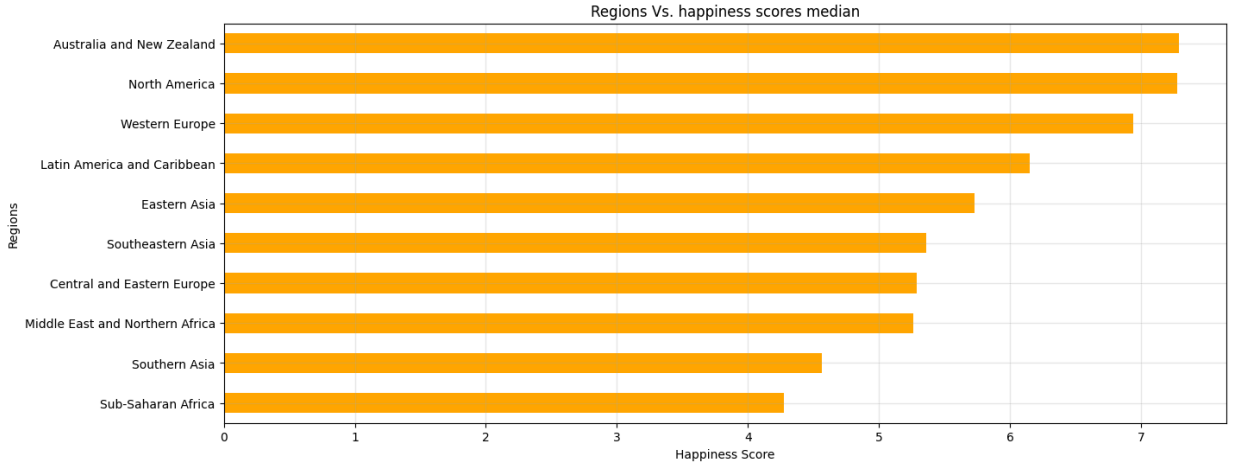
Out[64]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Tru (Governme Corruptio
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.0000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.1434
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.1200
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.0000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.0616
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.1072
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.1802
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.5519

It is noticed taht the happiness_rank column has a very high standard deviation of 45.182384.

The average happiness score is 5. 38, with a range of 2.69 to 7.77. Which shows quit a gap between the highest and lowest happiness score.

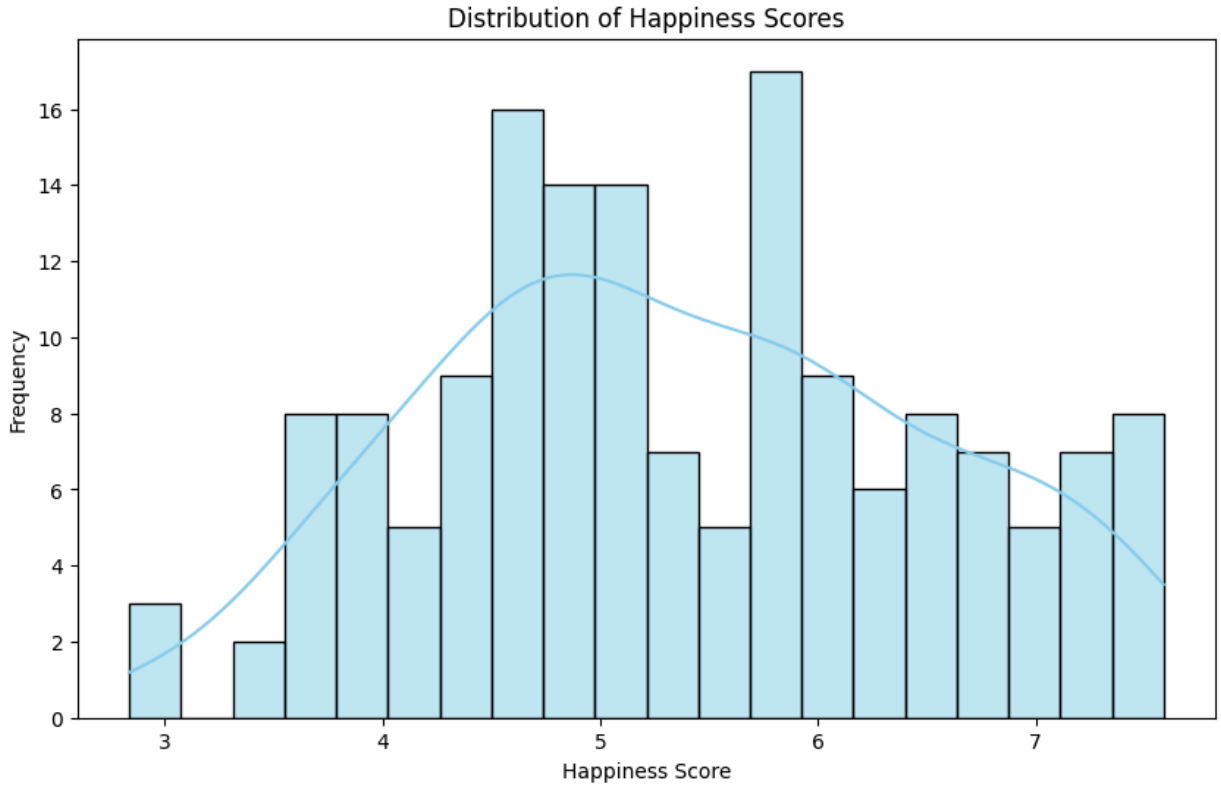
In [65]:
```
df.groupby('Region')['Happiness Score'].median().sort_values().plot(kind='barh', col
plt.title('Regions Vs. happiness scores median')
plt.xlabel('Happiness Score')
plt.ylabel('Regions');
plt.grid(True, alpha=0.3)
```



The region that has highest happiness score across all years is Australia and New Zealand.

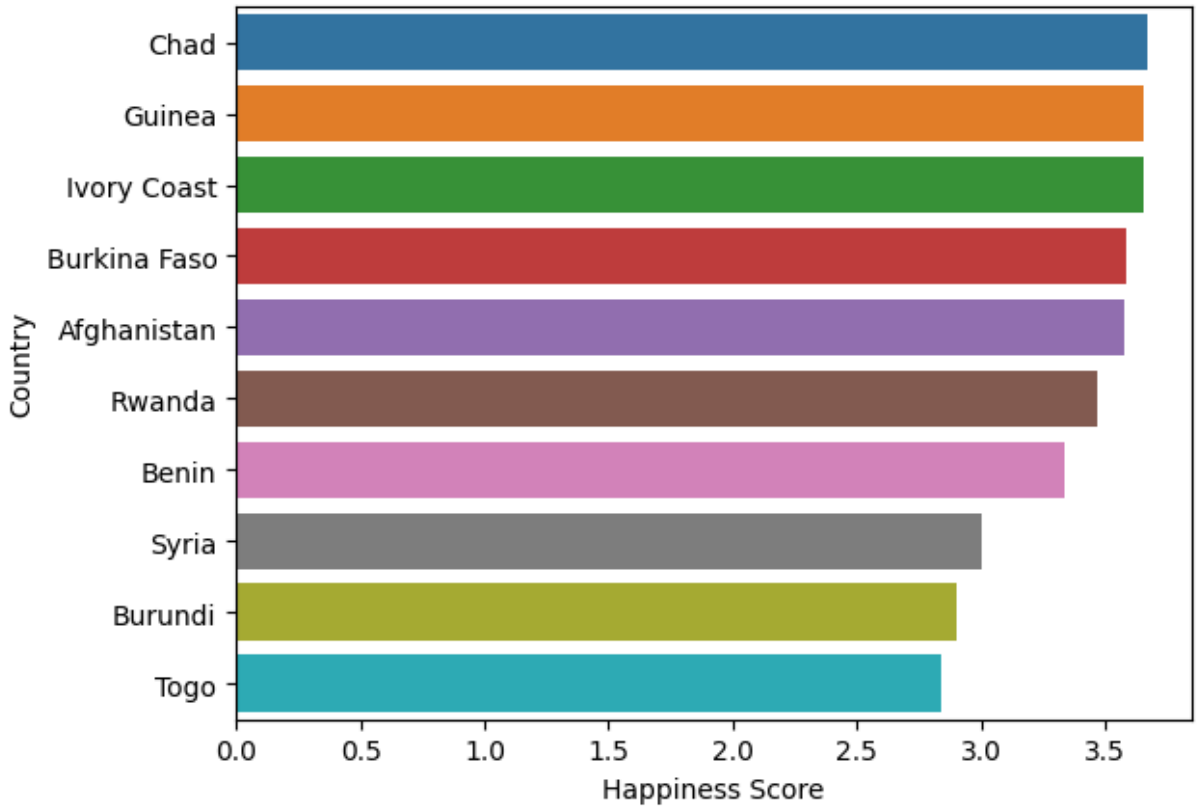
The region that has lowest happiness score across all years is Sub-Saharan Africa.

In [66]:
```
#Distribution of Happiness score
plt.figure(figsize=(10, 6))
sns.histplot(df['Happiness Score'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of Happiness Scores')
plt.xlabel('Happiness Score')
plt.ylabel('Frequency')
plt.show()
```



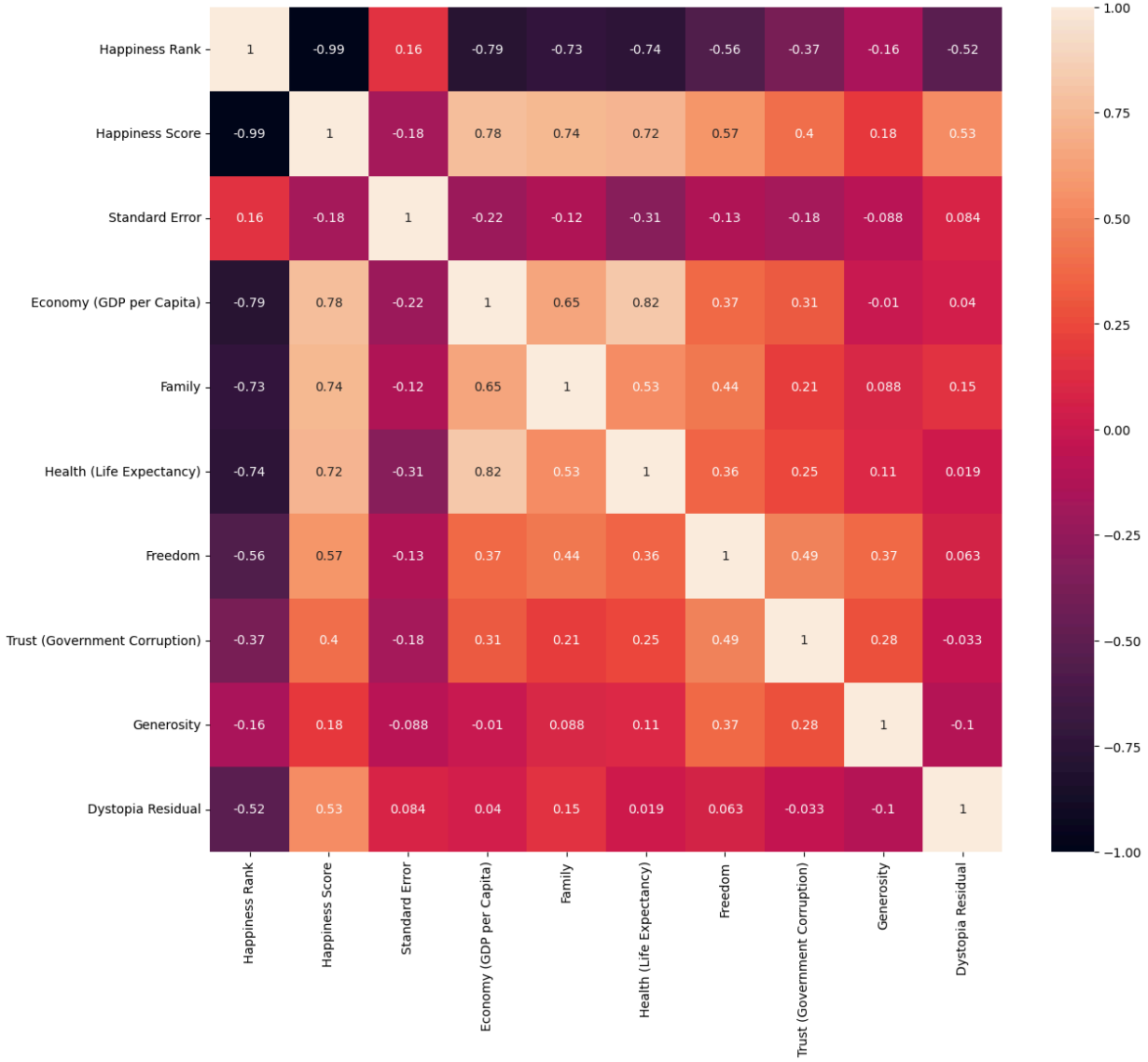
In [67]:
```
sns.barplot(y = df['Country'][-10:], x = df['Happiness Score'][-10:] )
```

Out[67]:
```
<Axes: xlabel='Happiness Score', ylabel='Country'>
```



In [68]:
```
plt.figure(figsize=(14,12))
sns.heatmap(df.corr(), annot=True, vmin=-1.0, vmax=1.0)
plt.show()
```

```
C:\Users\99Minds-1\AppData\Local\Temp\ipykernel_18484\392568953.py:2: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
ersion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, vmin=-1.0, vmax=1.0)
```



In [69]:
```
df.dtypes
```

Out[69]:
```
Country                           object
Region                            object
Happiness Rank                     int64
Happiness Score                  float64
Standard Error                   float64
Economy (GDP per Capita)         float64
Family                           float64
Health (Life Expectancy)         float64
Freedom                          float64
Trust (Government Corruption)    float64
Generosity                       float64
Dystopia Residual                float64
dtype: object
```

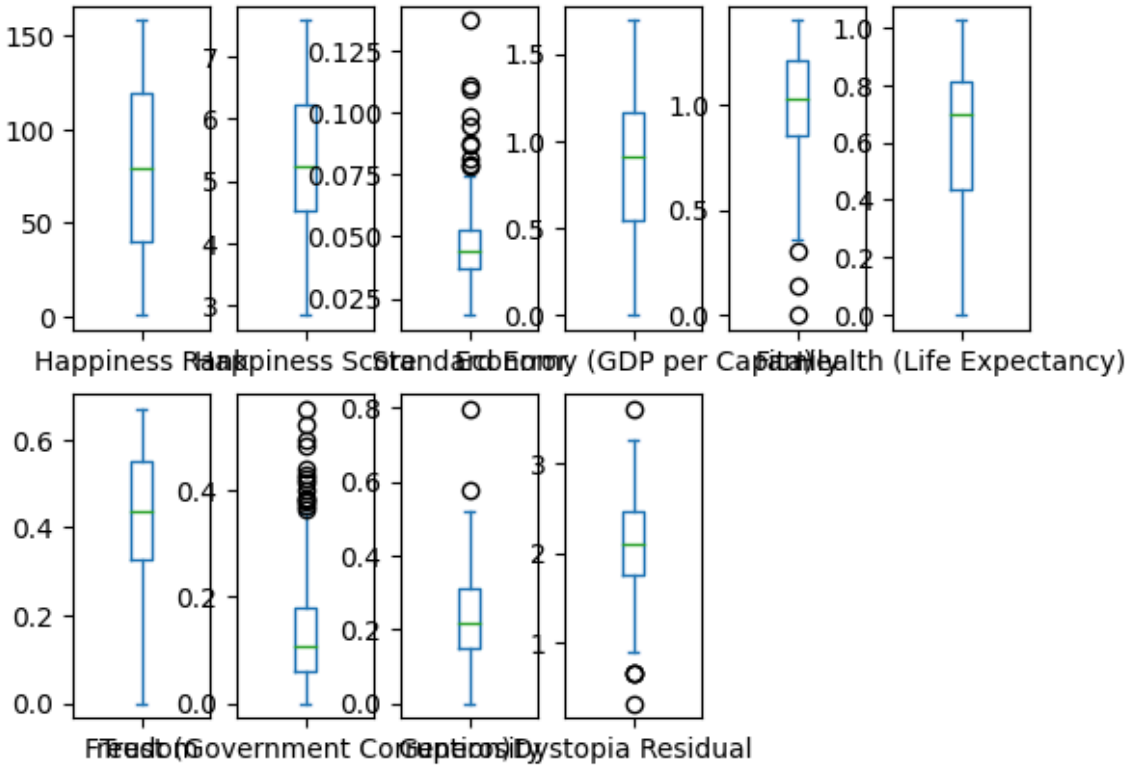
In [70]:
```
df.duplicated().any()
```

Out[70]:
```
False
```

There are no duplicates records found.

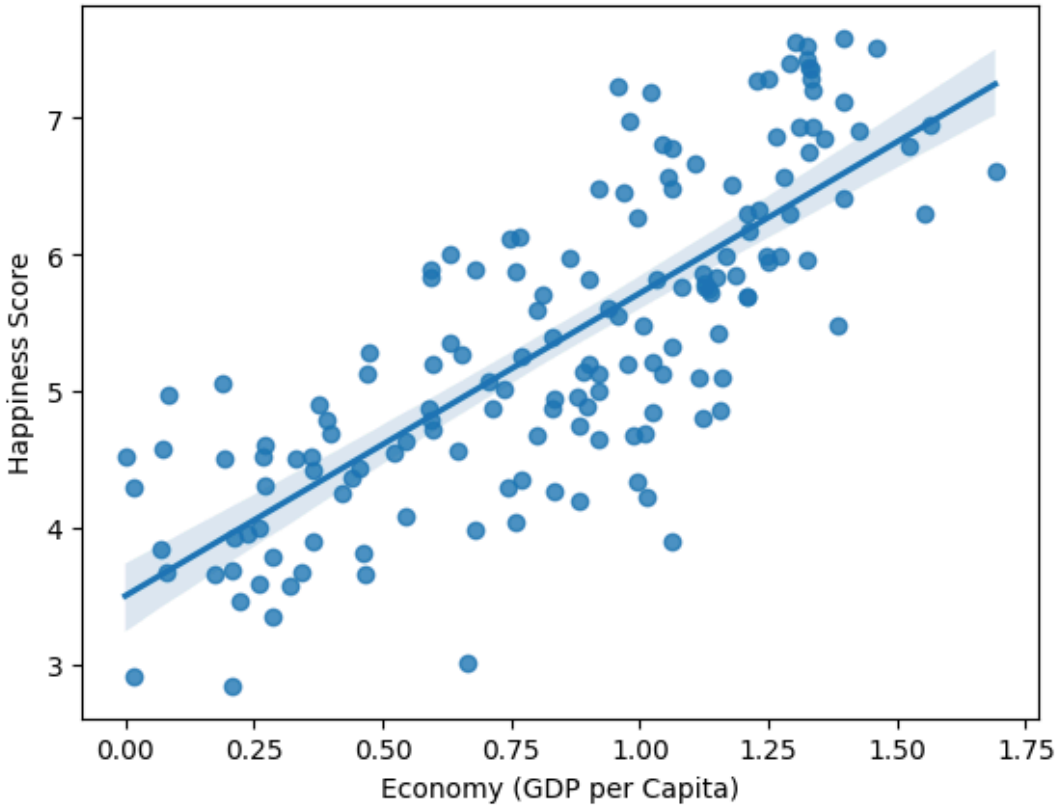
In [71]:
```
df.plot(kind='box',subplots=True, layout=(2,6))
```

Out[71]:
```
Happiness Rank                    Axes(0.125,0.53;0.110714x0.35)
Happiness Score                Axes(0.257857,0.53;0.110714x0.35)
Standard Error                 Axes(0.390714,0.53;0.110714x0.35)
Economy (GDP per Capita)       Axes(0.523571,0.53;0.110714x0.35)
Family                         Axes(0.656429,0.53;0.110714x0.35)
Health (Life Expectancy)       Axes(0.789286,0.53;0.110714x0.35)
Freedom                           Axes(0.125,0.11;0.110714x0.35)
Trust (Government Corruption)  Axes(0.257857,0.11;0.110714x0.35)
Generosity                     Axes(0.390714,0.11;0.110714x0.35)
Dystopia Residual              Axes(0.523571,0.11;0.110714x0.35)
dtype: object
```



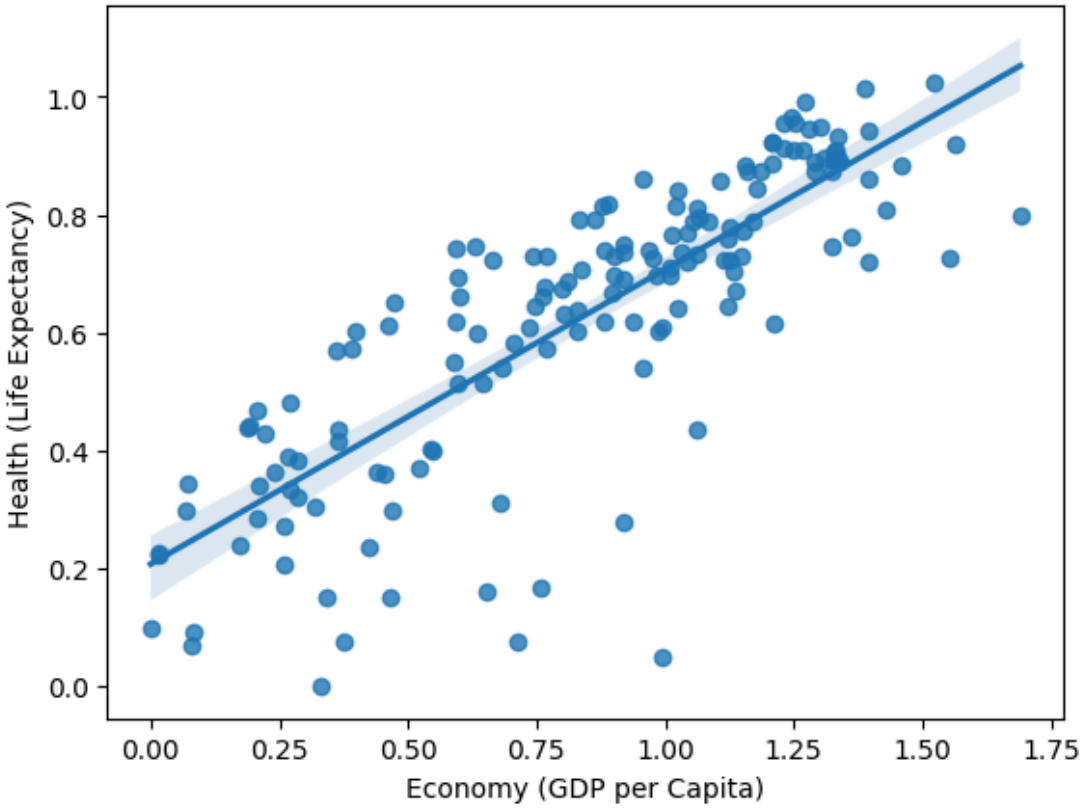
In [72]:
```
# Correlation between GDP per capita and  hapiness Score
sns.regplot(x='Economy (GDP per Capita)', y='Happiness Score', data=df)
```

Out[72]:
```
<Axes: xlabel='Economy (GDP per Capita)', ylabel='Happiness Score'>
```



In [73]:
```
# Correlation between GDP per capita and  hapiness Score
sns.regplot(x='Economy (GDP per Capita)', y='Health (Life Expectancy)', data=df)
```

Out[73]: <Axes: xlabel='Economy (GDP per Capita)', ylabel='Health (Life Expectancy)'>



In [74]:
```
df.drop(['Country', 'Region'], axis=1, inplace=True)
```

In [75]:
```
#splitting the data into x & y variable
X=df.drop('Happiness Score', axis=1)
y=df['Happiness Score']
```

In [76]:
```
scaler=MinMaxScaler()
X=pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

In [77]:
```
X
```

Out[77]:

	Happiness Rank	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystop Residu
0	0.000000	0.131954	0.826132	0.962403	0.918244	0.993789	0.760595	0.372895	0.6686
1	0.006369	0.256311	0.770412	1.000000	0.924496	0.938841	0.256292	0.548198	0.7250
2	0.012739	0.124947	0.784113	0.970297	0.853099	0.969615	0.876175	0.428947	0.6608
3	0.019108	0.171549	0.863099	0.949167	0.863409	1.000000	0.661394	0.435983	0.6527
4	0.025478	0.143943	0.784592	0.943219	0.883326	0.945112	0.597144	0.575602	0.6485
...	...	...	...	...	...	...	...	...	
153	0.974522	0.136429	0.131376	0.551764	0.418083	0.883953	1.000000	0.284314	0.1044
154	0.980892	0.152638	0.169573	0.252355	0.311241	0.723426	0.145132	0.229432	0.3985
155	0.987261	0.267370	0.392329	0.338668	0.704150	0.234184	0.342556	0.592790	0.0000
156	0.993631	0.574926	0.009051	0.296578	0.218444	0.176937	0.182312	0.247864	0.4595
157	1.000000	0.411904	0.123449	0.099805	0.277425	0.544294	0.194434	0.209592	0.3783

158 rows × 9 columns

In [78]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=4
```

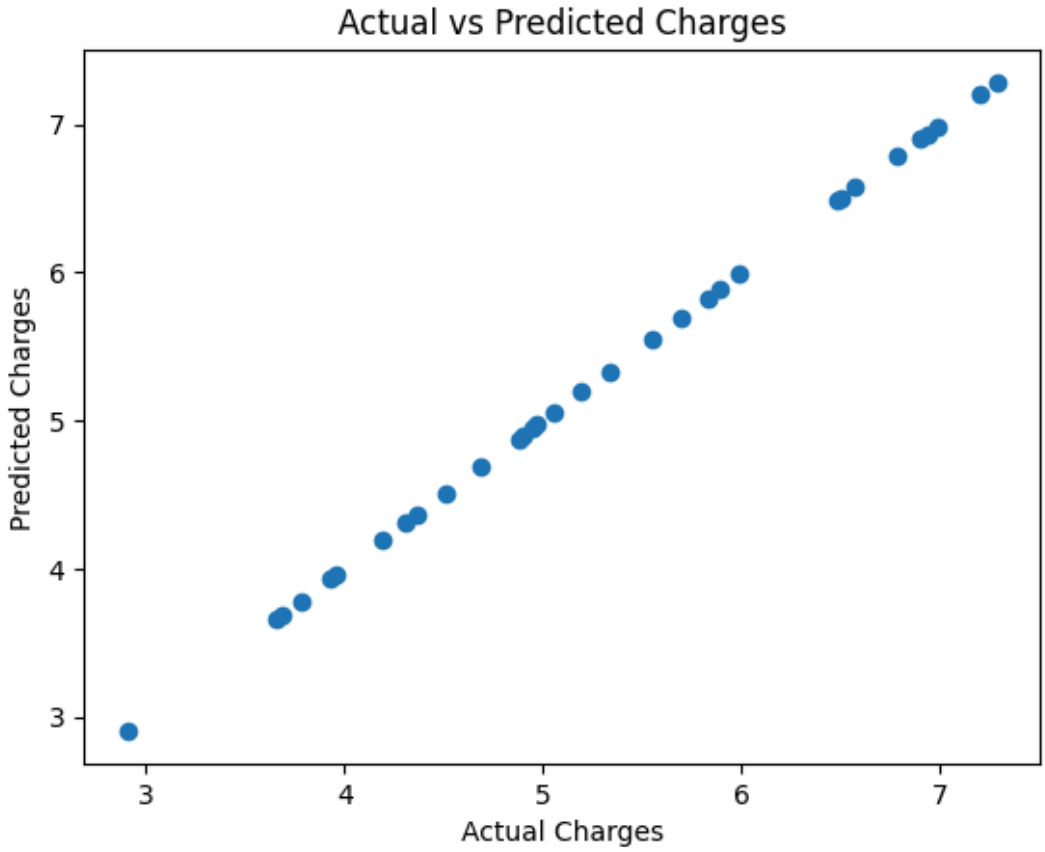
In [79]:
```
model=LinearRegression()
model.fit(X_train, y_train)
```

Out[79]:
```
LinearRegression
LinearRegression()
```

In [80]:
```
from sklearn.metrics import mean_absolute_error, r2_score

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Absolute Error: {mae}')
print(f'R2 Score: {r2}')
```

```
Mean Absolute Error: 0.00023374650842916678
R2 Score: 0.9999999476481373
```

In [81]:
```
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted Charges')
plt.show()
```



Thankyou

In [ ]:

In [ ]: