Project Name - **Happiness Score Prediction**

Name - Aman Mulla.

Batch - DS2307

Project Summary -

On March 20th, the International Day of Happiness is celebrated, accompanied by the release of the UN's World Happiness Report since 2017. This report ranks 155 countries based on their happiness levels, drawing attention from experts in economics, psychology, and foreign affairs. The ranking is derived from a survey using seven variables, measuring individuals' self-assessment of their well-being on a scale from 0 to 10. The scores are then normalized and compared to a hypothetical worst-case country called Dystopia.

While these variables may not entirely dictate a country's progress, they offer insights into what contributes to happiness. To understand the ranking better, a linear regression model is used to predict happiness scores, enabling a comparison between predicted and actual scores.

The report addresses questions and explores variations in these variables from 2015 to 2017. This analysis provides valuable insights into the correlation between these variables and happiness scores, aiding the comprehension of global well-being trends.

We have below feature in dataset,

1. **Country:** The name of the country being evaluated.
2. **Region:** The geographical region to which the country belongs.
3. **Happiness Rank:** The country's position in the World Happiness Report's ranking based on happiness scores.
4. **Happiness Score:** A measure of a country's overall happiness and well-being.
5. **Standard Error:** The level of uncertainty or margin of error associated with the happiness score.
6. **Ecomany (GDP per capita):** The country's economic strength, typically measured by Gross Domestic Product (GDP) per capita.
7. **Family:** The strength and quality of social support and family relationships within a country.
8. **Health(Life Expectancy):** The average life expectancy within the country, reflecting the health and healthcare system.
9. **Freedom:** The degree of individual freedom and personal liberties in a country.
10. **Trust(Government Corruption):** A measure of trust in the government and the absence of corruption within a country.
11. **Generosity:** A measure of the willingness to help and support others within a country.
12. **Dystopia Residual:** A hypothetical benchmark representing the lowest possible well-being, used for comparison in the happiness ranking.

***Dystopia is an imaginary country that has the world's least-happy people. The purpose in establishing Dystopia is to have a benchmark against which all countries can be favorably compared (no country performs more poorly than Dystopia) in terms of each of the six key variables, thus allowing each sub-bar to be of positive width. The lowest scores observed for the six key variables, therefore, characterize Dystopia. Since life would be very unpleasant in a country with the world's lowest incomes, lowest life expectancy, lowest generosity, most corruption, least freedom and least social support, it is referred to as "Dystopia," in contrast to Utopia.

***The residuals, or unexplained components, differ for each country, reflecting the extent to which the six variables either over- or under-explain average life evaluations. These residuals have an average value of approximately zero over the whole set of countries.

Problem Statement

Develop a predictive model to analyze and understand the factors influencing a country's happiness, as outlined in the World Happiness Report. This project aims to address the question of how various socio-economic and societal factors, such as GDP per capita, social support, health, freedom, trust in government, and generosity, impact a country's happiness score.

The model should assess the relative importance of these variables and their changes over a specified time period (e.g., 2015-2017). Additionally, the project will investigate whether these factors differ among regions and identify any significant correlations between them.

Knowing data and variable in dataset

```
# Importing Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option('display.max_rows', None)
```

```
happiness_data = pd.read_csv('/content/drive/MyDrive/DataSets/happiness_score_dataset11.csv')

happiness_data.head()
```

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176

```
# Check for shape of dataset

happiness_data.shape
```

```
(158, 12)
```

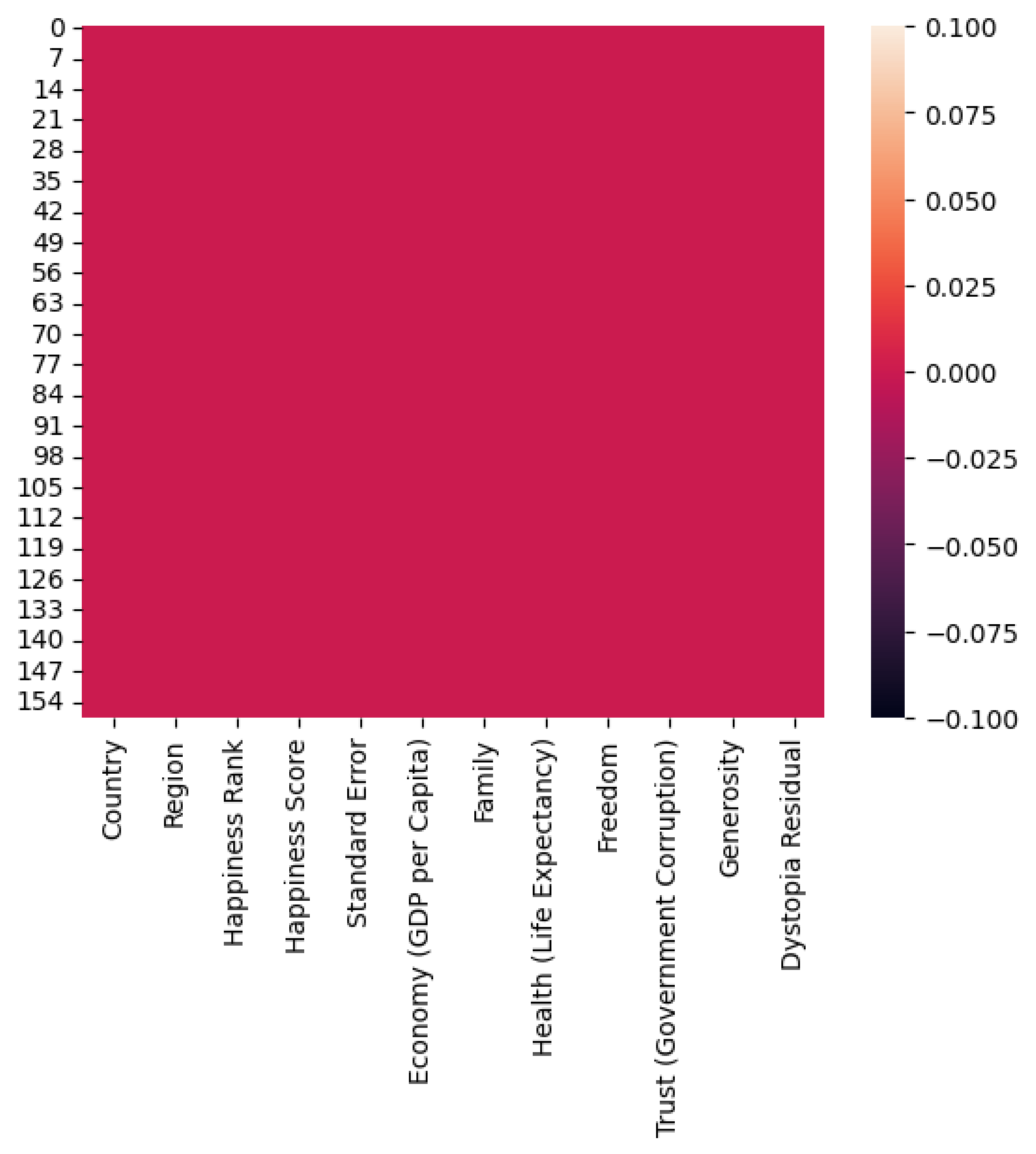
For our dataset we have total 158 rows and 12 columns.

```
# will check for missing values in our dataset

happiness_data.isnull().sum()

happiness_data.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12     False
13     False
14     False
15     False
16     False
17     False
18     False
19     False
20     False
21     False
22     False
23     False
24     False
25     False
26     False
27     False
28     False
29     False
30     False
31     False
32     False
33     False
34     False
35     False
36     False
37     False
38     False
39     False
40     False
41     False
42     False
43     False
44     False
45     False
46     False
47     False
48     False
49     False
50     False
51     False
52     False
53     False
54     False
55     False
56     False
57     False
```

```
sns.heatmap(happiness_data.isnull())
```

```
<Axes: >
```



From above we can check that there is no any null values and no any duplicate value present in the dataset.

To avoid type error will remane for some of column names.

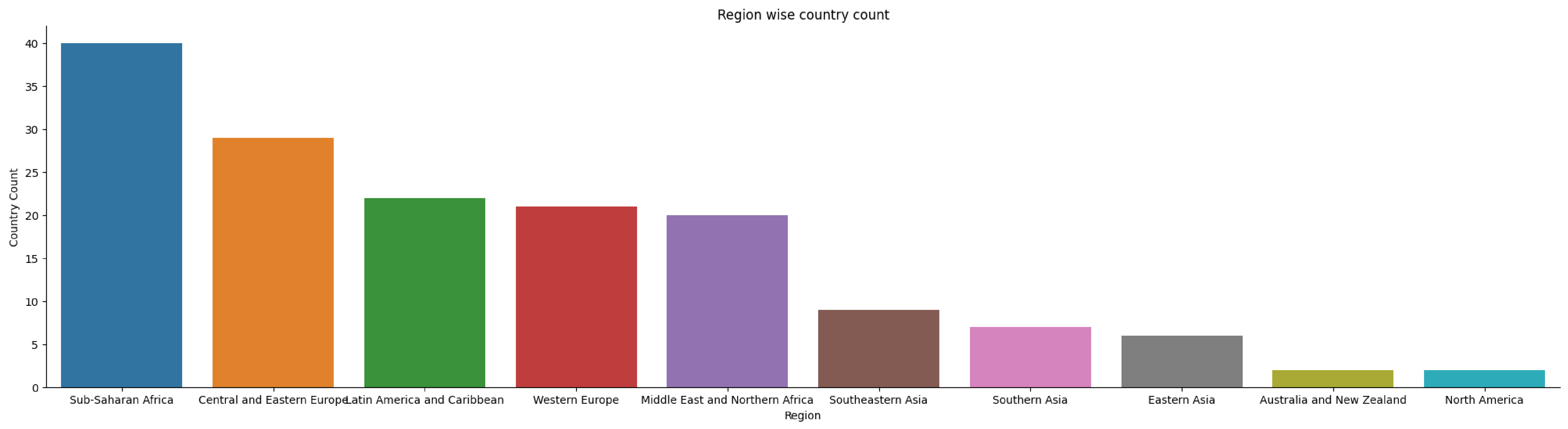
```
happiness_data.rename(columns={'Happiness Rank': 'Happiness_Rank'}, inplace=True)

happiness_data.rename(columns={'Happiness Score': 'Happiness_Score'}, inplace=True)

happiness_data.rename(columns={'Standard Error': 'Standard_Error'}, inplace=True)

happiness_data.rename(columns={'Economy (GDP per Capita)': 'Economy_(GDP_per_Capita)'}, inplace=True)

happiness_data.rename(columns={'Health (Life Expectancy)': 'Health_(Life_Expectancy)'}, inplace=True)

happiness_data.rename(columns={'Trust (Government Corruption)': 'Trust_Government_Corruption'}, inplace=True)

happiness_data.rename(columns={'Dystopia Residual': 'Dystopia_Residual'}, inplace=True)
```

Chart - 1

Region wise country count

```
region_country = happiness_data.groupby('Region')['Country'].count().sort_values(ascending=False)

print(region_country)


f,ax = plt.subplots(figsize=(25,6))
sns.despine(f)
sns.barplot(x=region_country.index, y=region_country.values, ax=ax)
plt.xlabel('Region')
plt.ylabel('Country Count')
plt.title('Region wise country count')
plt.show()
```

```
Region
Sub-Saharan Africa                 40
Central and Eastern Europe         29
Latin America and Caribbean        22
Western Europe                     21
Middle East and Northern Africa    20
Southeastern Asia                   9
Southern Asia                       7
Eastern Asia                        6
Australia and New Zealand           2
North America                       2
Name: Country, dtype: int64
```



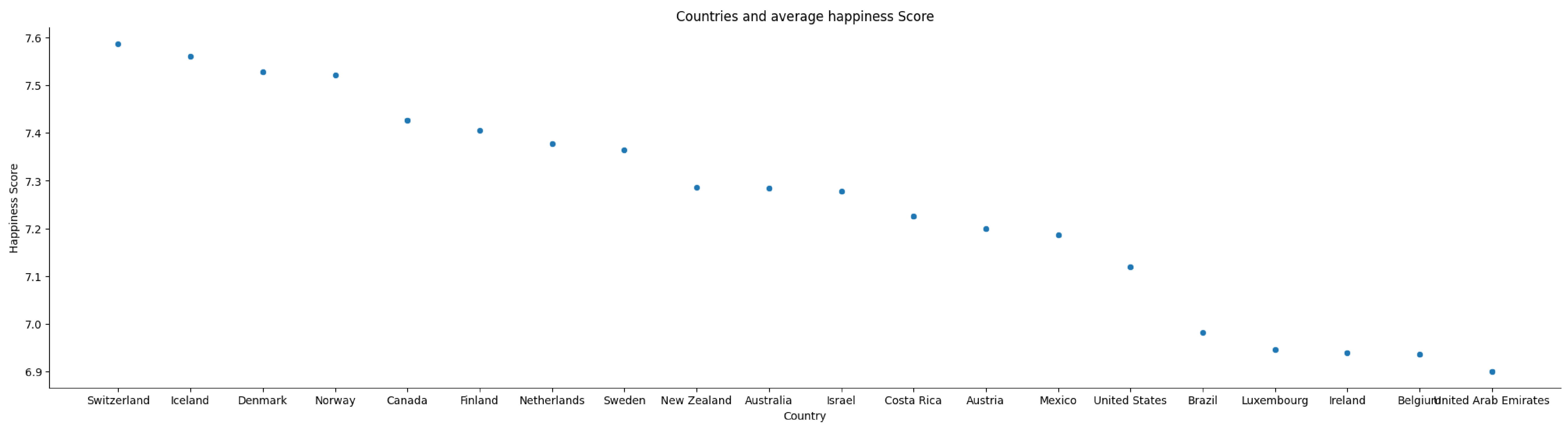
Insights from above chart:

1. From above chart we can observe that we have region wis country count, in which 'Sub-Saharan Africa' have highest number of countries while 'North America' hava less number of countries.

Chart - 2

Countries and average happiness Score

```
counties_happinessScore = happiness_data.groupby('Country')['Happiness_Score'].mean().sort_values(ascending=False).head(20)

counties_happinessScore


f,ax = plt.subplots(figsize=(25,6))
sns.despine(f)
sns.scatterplot(x=counties_happinessScore.index,y=counties_happinessScore.values,data=counties_happinessScore)
plt.xlabel('Country')
plt.ylabel('Happiness Score')
plt.title('Countries and average happiness Score')
plt.show()
```

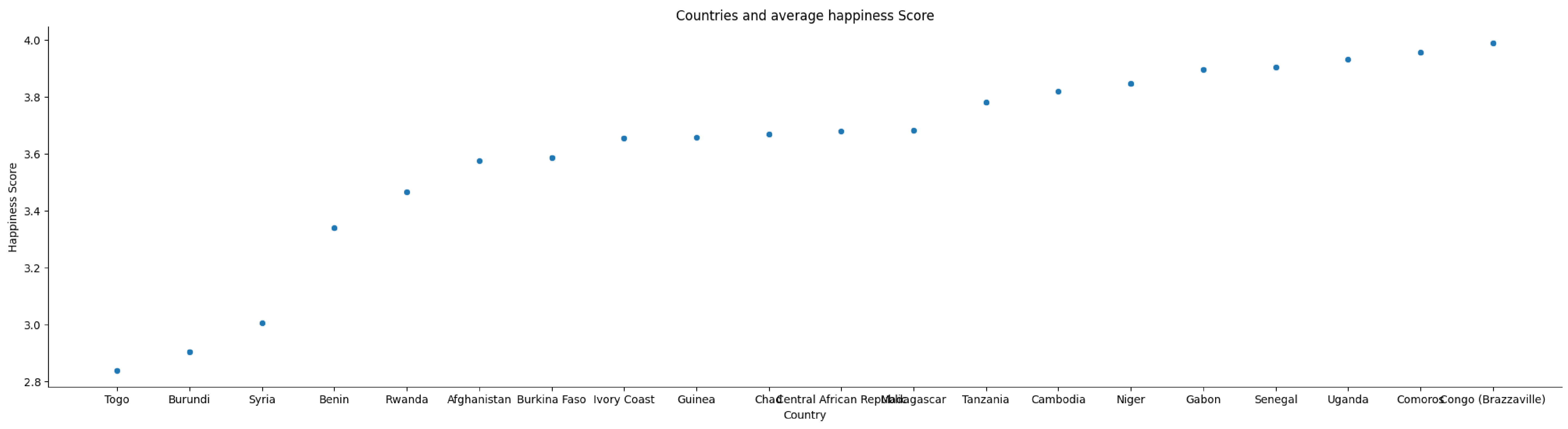


Insights from above chart:

1. From abve scatter plot we can observe that **Switzerland** is having higest happiness score among top 20 counries while, **Arab Emirates** is lowest happiness score in top 20 countries.

```
counties_happinessScore = happiness_data.groupby('Country')['Happiness_Score'].mean().sort_values(ascending=True).head(20)

counties_happinessScore
```

```
f,ax = plt.subplots(figsize=(25,6))
sns.despine(f)
sns.scatterplot(x=counties_happinessScore.index,y=counties_happinessScore.values,data=counties_happinessScore)
plt.xlabel('Country')
plt.ylabel('Happiness Score')
plt.title('Countries and average happiness Score')
plt.show()
```



Insights from above chart:

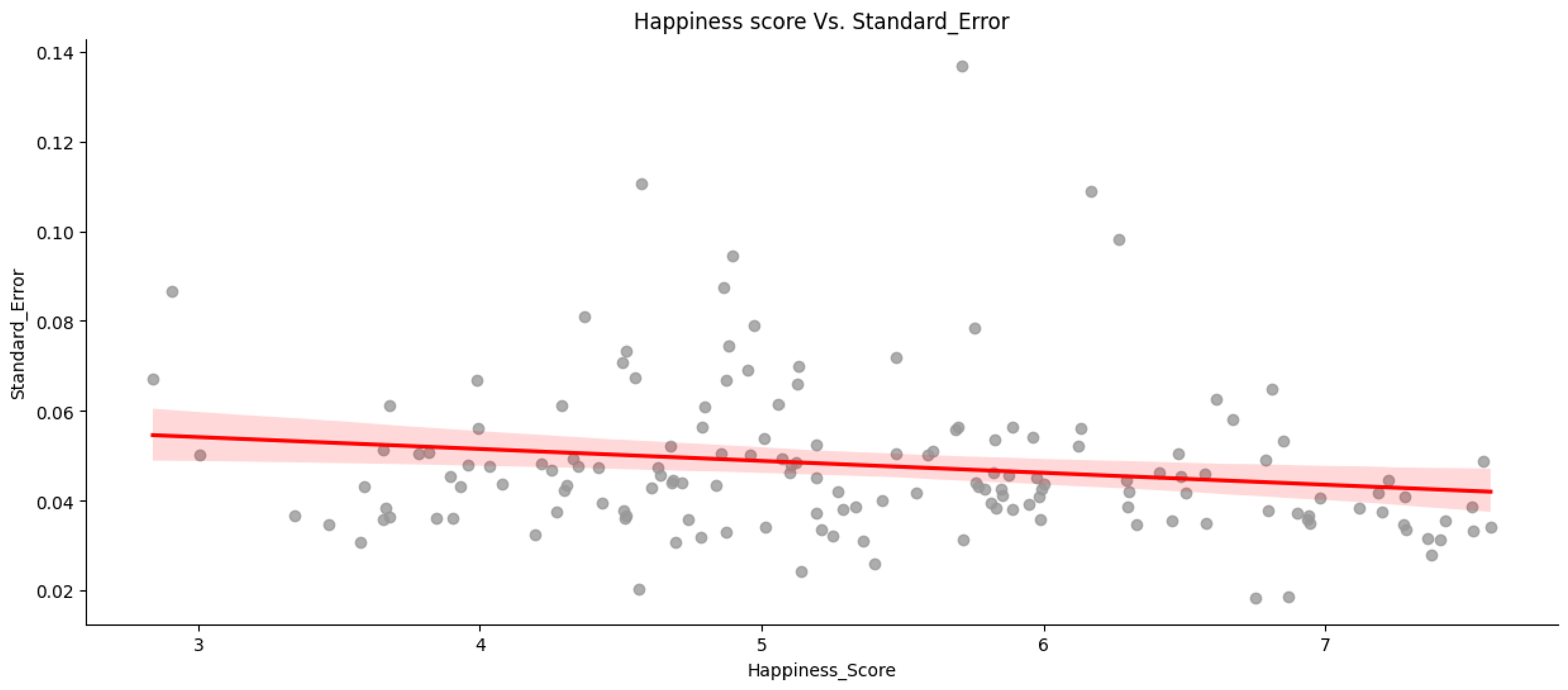
1. From above chart we can observe that **Togo** is least happiness score country in total countries.

Now will check relation of eah factor with Happiness score.

Chart - 3

Happiness score Vs. Standard_Error

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Standard_Error', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Standard_Error')
plt.title('Happiness score Vs. Standard_Error')
plt.show()
```



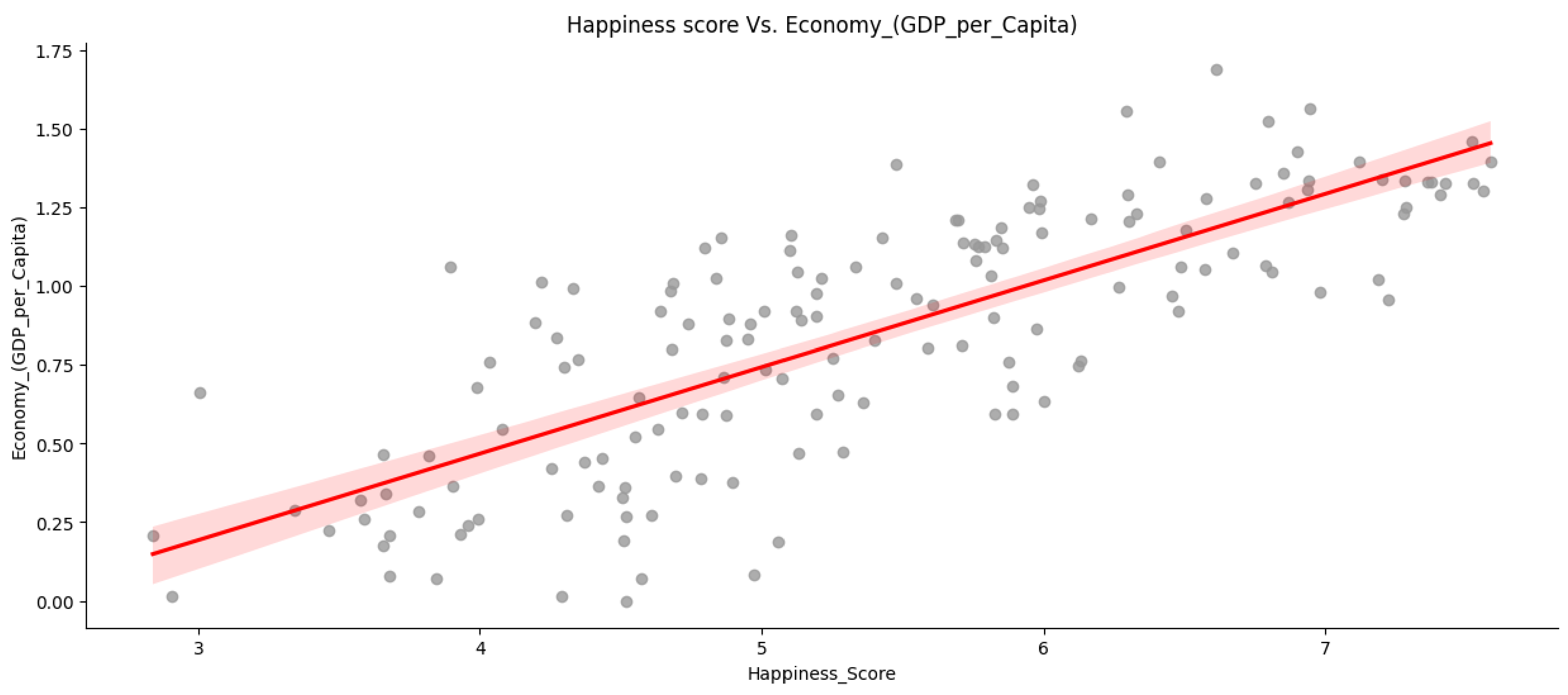
Insights from above chart:

1. The regression line shows a slight negative correlation between 'Happiness_Score' and 'Standard_Error.' As 'Happiness_Score' increases, 'Standard_Error' tends to decrease. This suggests that countries with higher happiness scores tend to have more precise or lower standard errors in their measurements.

Chart - 4

Happiness score Vs. Economy_(GDP_per_Capita)

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Economy_(GDP_per_Capita)', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Economy_(GDP_per_Capita)')
plt.title('Happiness score Vs. Economy_(GDP_per_Capita)')
plt.show()
```



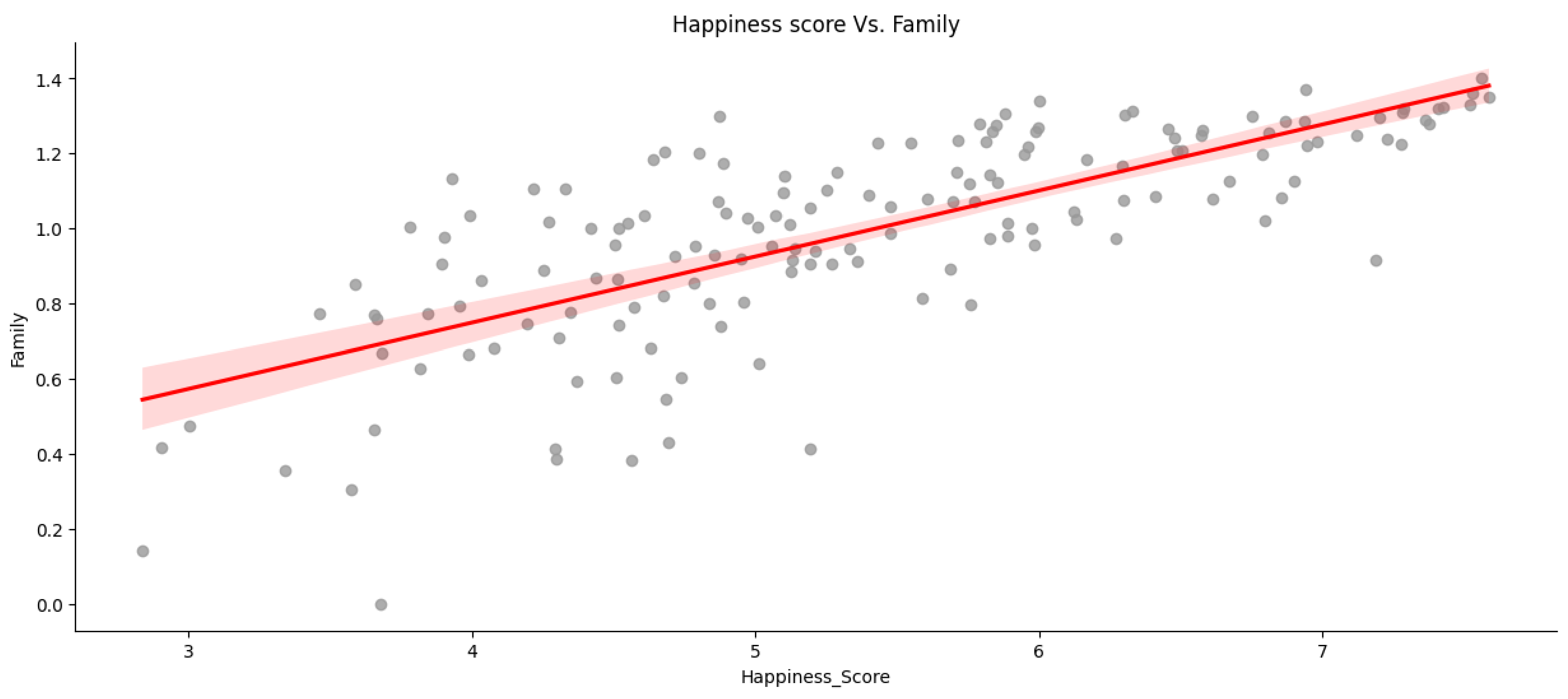
Insights from above chart:

1. The plot displays a positive correlation between 'Happiness_Score' and 'Economy_(GDP_per_Capita).' As the 'Economy_(GDP_per_Capita)' value increases, the 'Happiness_Score' tends to increase as well. This suggests that there is a tendency for people in countries with higher GDP per capita to report higher levels of happiness.**

Chart - 5

Happiness score Vs. Family

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Family', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Family')
plt.title('Happiness score Vs. Family')
plt.show()
```



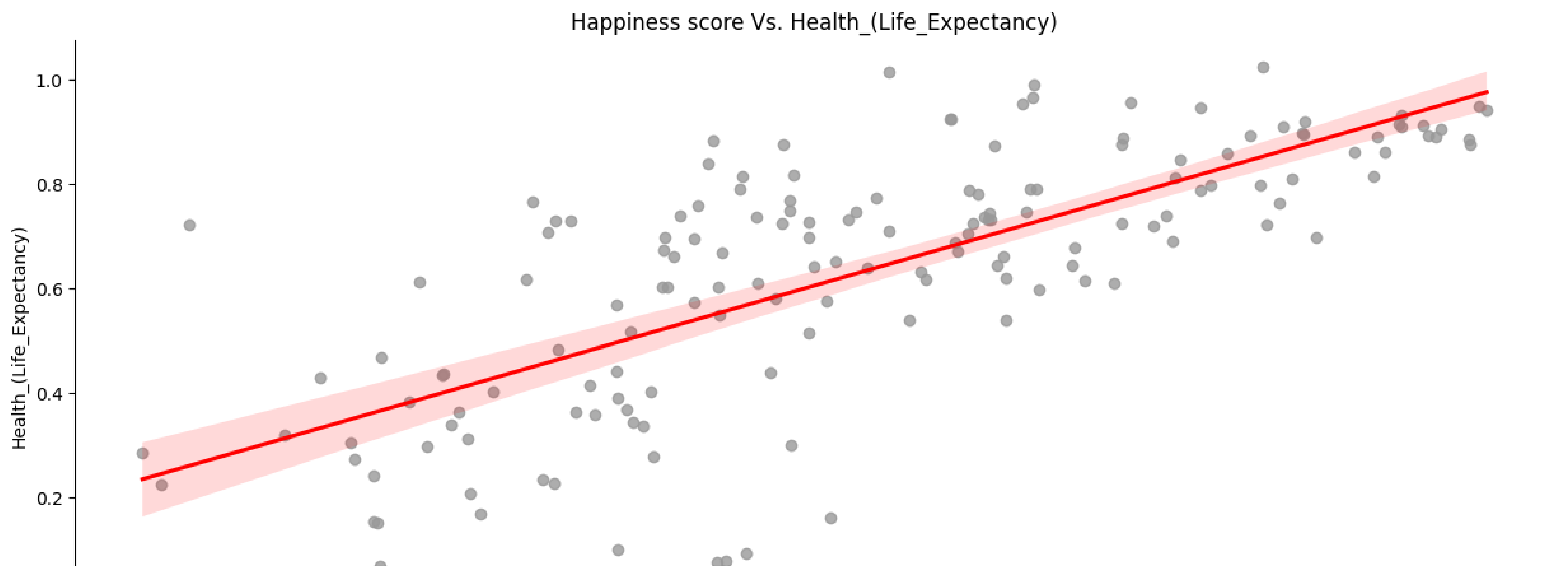
Insights from above chart:

1. The regression line has a positive slope, indicating a positive relationship between "Happiness_Score" and "Family." This means that as the "Happiness_Score" increases, the average "Family" score also tends to increase.There is a positive relationship between a country's "Happiness_Score" and the strength of the "Family" unit within that country.**

Chart - 6

Happiness score Vs. Health_(Life_Expectancy)

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Health_(Life_Expectancy)', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Health_(Life_Expectancy)')
plt.title('Happiness score Vs. Health_(Life_Expectancy)')
plt.show()
```



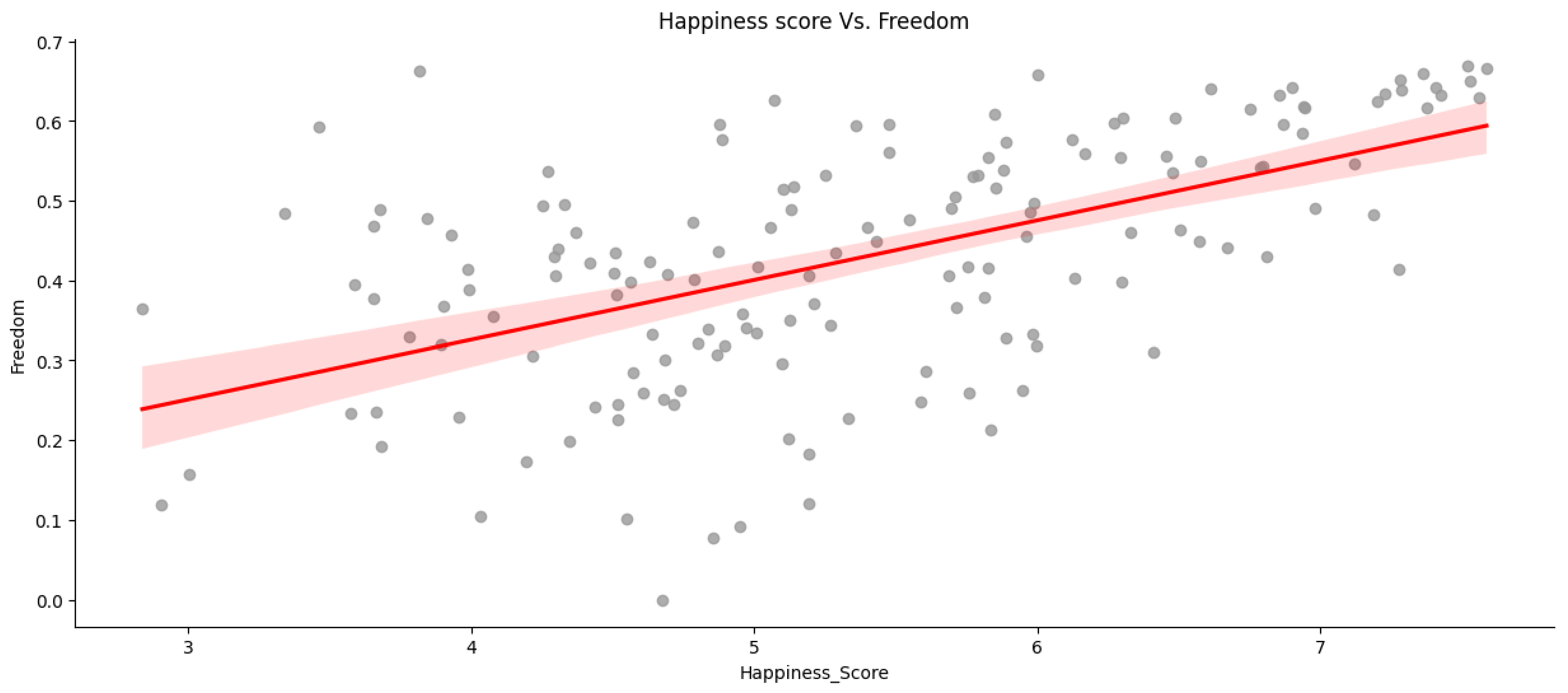
Insights from above chart:

1. The trend line, in this case, has a positive slope, which suggests a positive correlation between "Happiness_Score" and "Health_(Life_Expectancy)." In other words, as the happiness score increases, the life expectancy tends to increase as well.**

Chart - 7

Happiness score Vs. Freedom

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Freedom', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Freedom')
plt.title('Happiness score Vs. Freedom')
plt.show()
```



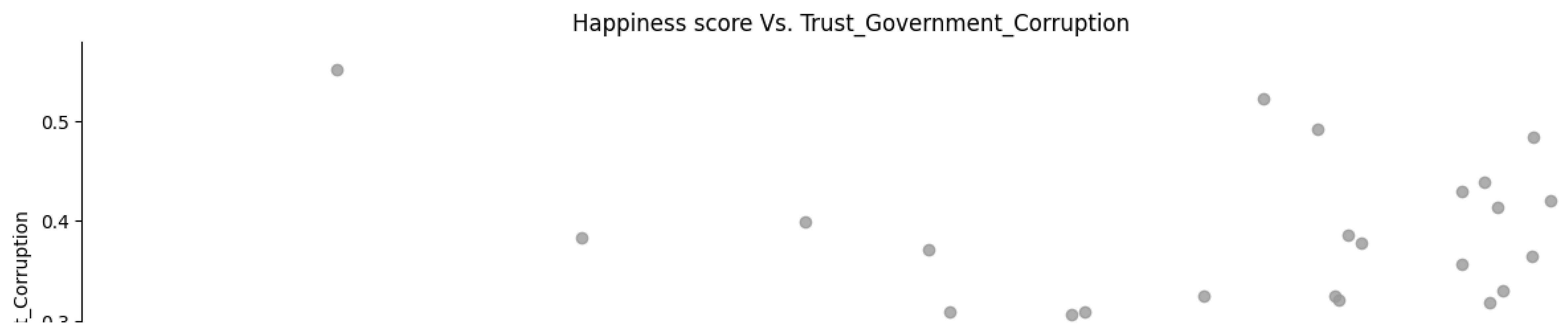
Insights from above chart:

1. The red regression line represents the overall trend in the data. It shows the general direction and strength of the relationship between 'Happiness_Score' and 'Freedom'. In this case, it appears that there is a positive linear relationship, suggesting that as 'Happiness_Score' increases, 'Freedom' tends to increase as well.**

Chart - 8

Happiness score Vs. Trust_(Government_Corruption)

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Trust_Government_Corruption', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Trust_Government_Corruption')
plt.title('Happiness score Vs. Trust_Government_Corruption')
plt.show()
```



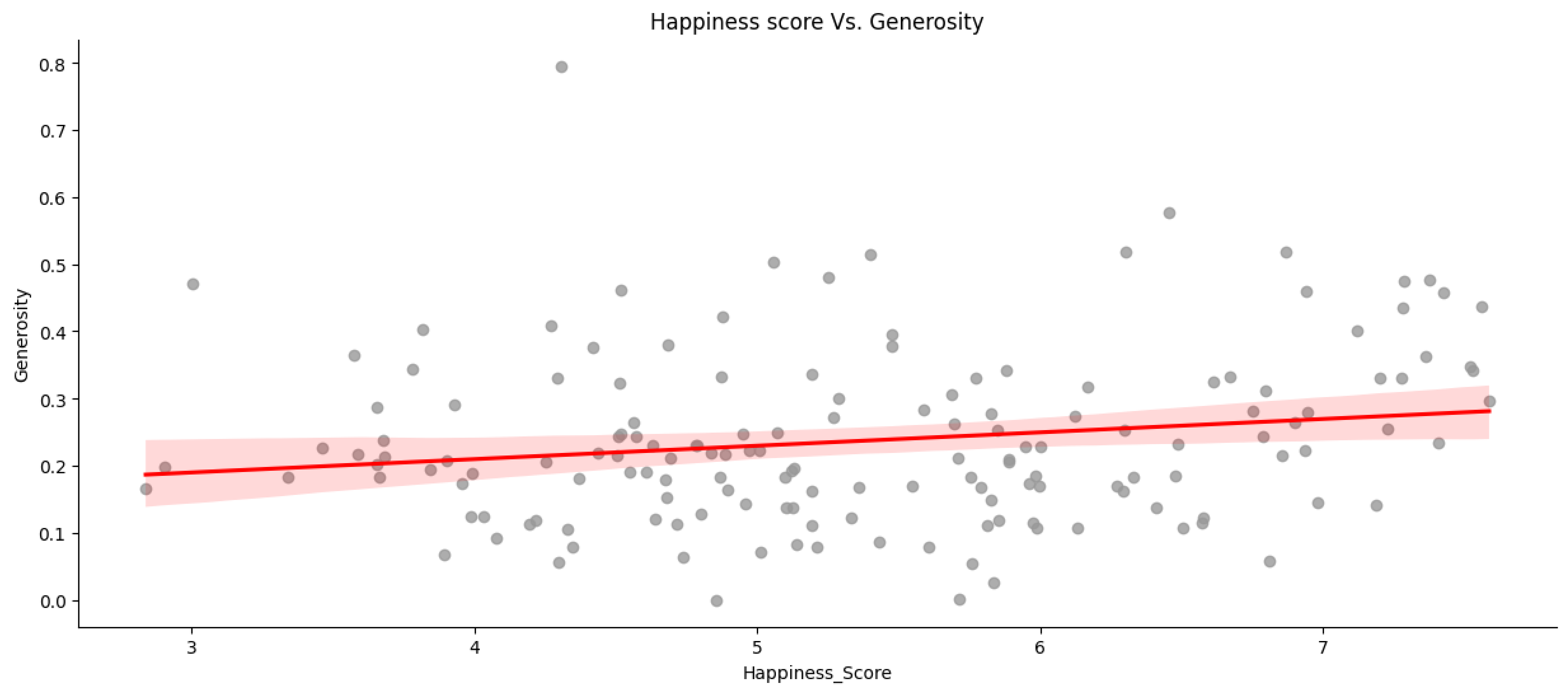
Insights from above chart:

1. The scatter plot shows that there is a general trend where as the 'Happiness_Score' increases, 'Trust_(Government_Corruption)' tends to increase as well. In other words, there appears to be a positive correlation between happiness scores and trust in government.**

Chart - 9

Happiness score Vs. Generosity

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Generosity', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Generosity')
plt.title('Happiness score Vs. Generosity')
plt.show()
```



Insights from above chart:

1. The regression line shows a slight positive correlation between 'Happiness_Score' and 'Generosity.' As 'Happiness_Score' increases, 'Generosity' tends to slightly increases. This suggests that countries with higher happiness scores tend to have more precise or slightly higher increases in their measurements.**

Chart - 10

Happiness score Vs. Dystopia_Residual

```
f,ax = plt.subplots(figsize=(15,6))
sns.despine(f)
sns.regplot(data=happiness_data, x='Happiness_Score', y='Dystopia_Residual', scatter=True, color=".6", line_kws=dict(color="r"))
plt.xlabel('Happiness_Score')
plt.ylabel('Dystopia_Residual')
plt.title('Happiness score Vs. Dystopia_Residual')
plt.show()
```

Insights from above chart:

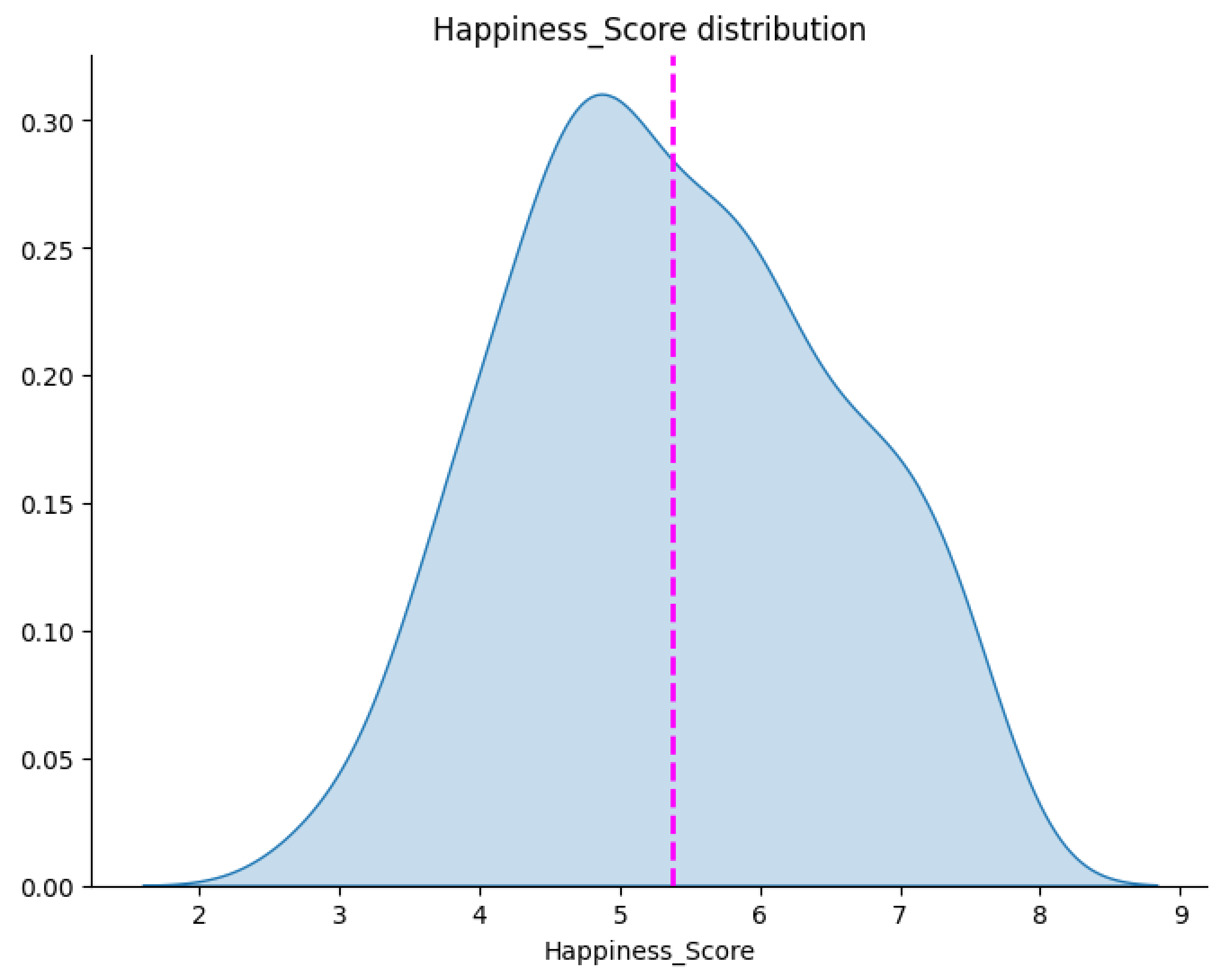
1. The trendline in the plot shows a positive correlation between 'Happiness_Score' and 'Dystopia_Residual.' As 'Happiness_Score' increases, 'Dystopia_Residual' tends to increase as well. This suggests that countries with higher happiness scores tend to have higher dystopia residual values.**

Chart - 11

Happiness score distribution plot

```
f,ax  = plt.subplots(figsize=(8,6))

sns.despine(f)
dist = sns.kdeplot(happiness_data['Happiness_Score'], shade=True)
dist.set(xlabel = 'Happiness_Score', ylabel ='', title = 'Happiness_Score distribution')

dist.axvline(happiness_data['Happiness_Score'].mean(), color='magenta', linestyle='dashed', linewidth=2)

plt.show()
```

```
<ipython-input-20-772f89a5c756>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  dist = sns.kdeplot(happiness_data['Happiness_Score'], shade=True)
```



Insights from above chart:

1. The KDE plot shows the shape of the 'Happiness_Score' distribution. In this case, it appears to be somewhat skewed to the right (positively skewed), indicating that there might be more countries with higher happiness scores compared to lower scores.
2. Dashed line represents the mean of the 'Happiness_Score.' It indicates the central tendency of the distribution which is around 5.3. In this plot, the mean score appears to be around the value where the peak of the KDE plot is.

From above all chart we can observe that only Standard Error variable is having negaive correlation with Happiness Score, while all other is having positive corrlation.

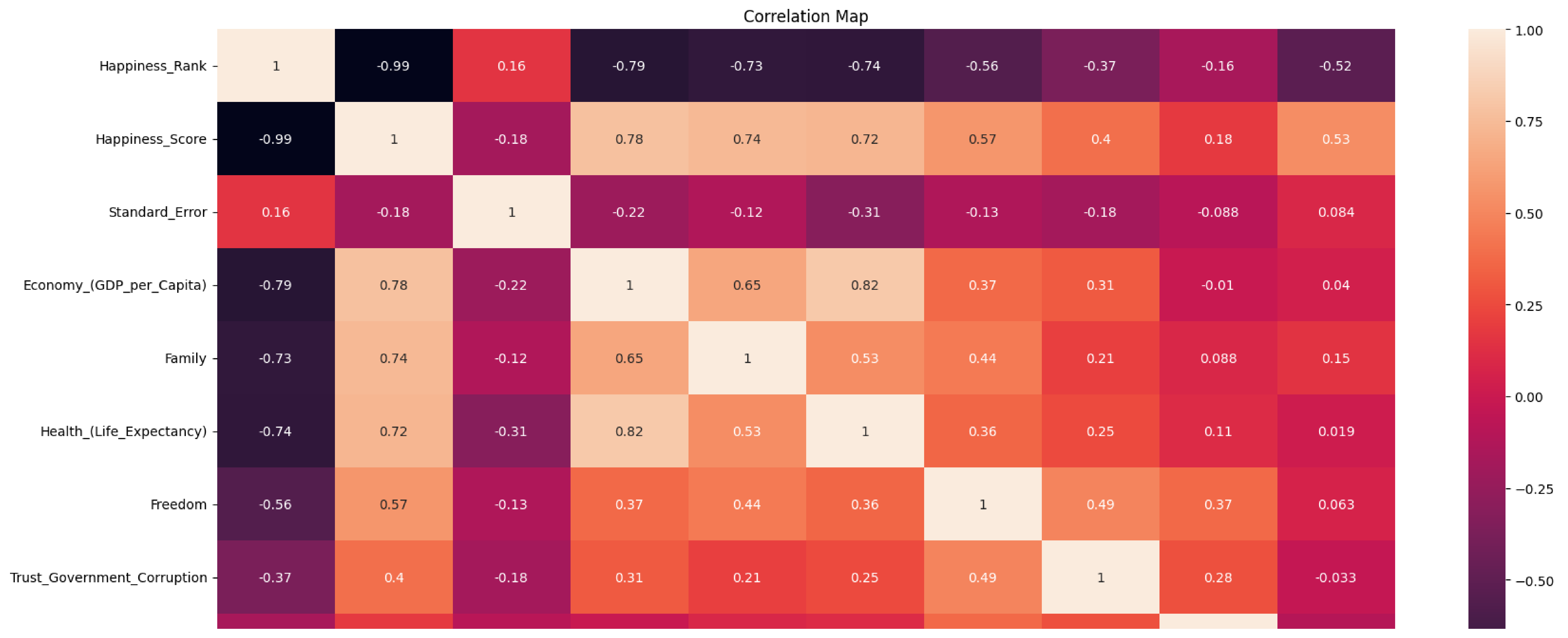
Will Check for heatmap for more clarrification on correlations.

Chart - 12

Heatmap

```
correlation_data = happiness_data

correlation_matrix = correlation_data.corr()

plt.figure(figsize=(20,10))

sns.heatmap(correlation_matrix,annot=True)
plt.title('Correlation Map')
plt.show()
```

```
<ipython-input-21-e9b7a624e6f2>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid column
  correlation_matrix = correlation_data.corr()
```



Happiness_Rank	1	-0.99	0.16	-0.79	-0.73	-0.74	-0.56	-0.37	-0.16	-0.52
Happiness_Score	-0.99	1	-0.18	0.78	0.74	0.72	0.57	0.4	0.18	0.53
Standard_Error	0.16	-0.18	1	-0.22	-0.12	-0.31	-0.13	-0.18	-0.088	0.084
Economy_(GDP_per_Capita)	-0.79	0.78	-0.22	1	0.65	0.82	0.37	0.31	-0.01	0.04
Family	-0.73	0.74	-0.12	0.65	1	0.53	0.44	0.21	0.088	0.15
Health_(Life_Expectancy)	-0.74	0.72	-0.31	0.82	0.53	1	0.36	0.25	0.11	0.019
Freedom	-0.56	0.57	-0.13	0.37	0.44	0.36	1	0.49	0.37	0.063
Trust_Government_Corruption	-0.37	0.4	-0.18	0.31	0.21	0.25	0.49	1	0.28	-0.033

Insights from above chart:

1. From above heatmap we can observe that,
2. Positive correlation : 'Happiness_Score' and 'Economy_(GDP_per_Capita)' ,'Happiness_Score' and 'Family' , 'Happiness_Score' and 'Health_(Life_Expectancy)' , 'Health_(Life_Expectancy)' and 'Economy_(GDP_per_Capita)'.
3. Negative correlation : 'Happiness_Score' and 'Standard_Error' , 'Standard_Error' and 'Economy_(GDP_per_Capita)' , 'Standard_Error' and 'Family' , 'Standard_Error' and 'Freedom'.

Will check for Variance Inflation Factor to find out multicolinearity between variables.

```
# to get VIF will define x and y

x = happiness_data.drop(columns=['Happiness_Rank','Happiness_Score','Country','Region'])
y = happiness_data['Happiness_Score']

from sklearn.preprocessing import StandardScaler


scalar = StandardScaler()

x_scaled =scalar.fit_transform(x)

vif = pd.DataFrame()

vif['vif'] = [variance_inflation_factor(x_scaled,i) for i in range(x_scaled.shape[1])]

vif['features'] = x.columns

vif
```

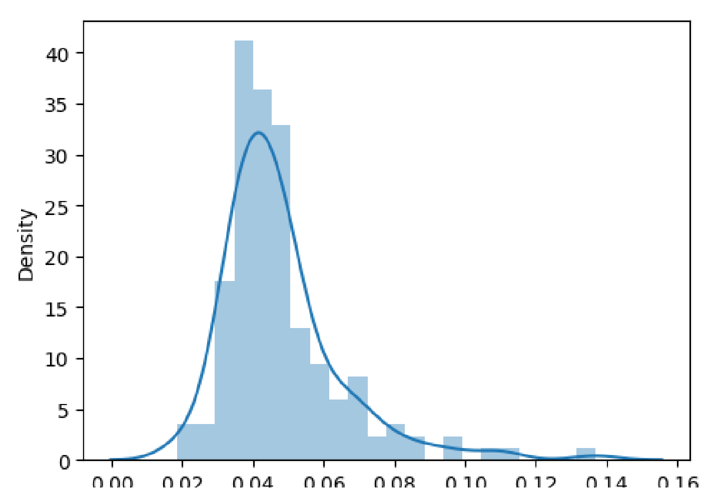
From above VIF score we can observe that, Non of variables VIF is more than 5, hence seems like no any multicollinearity. But at this dataset is less and short in nature will consider 'Economy_(GDP_per_Capita)' variable is having multicollinearity.

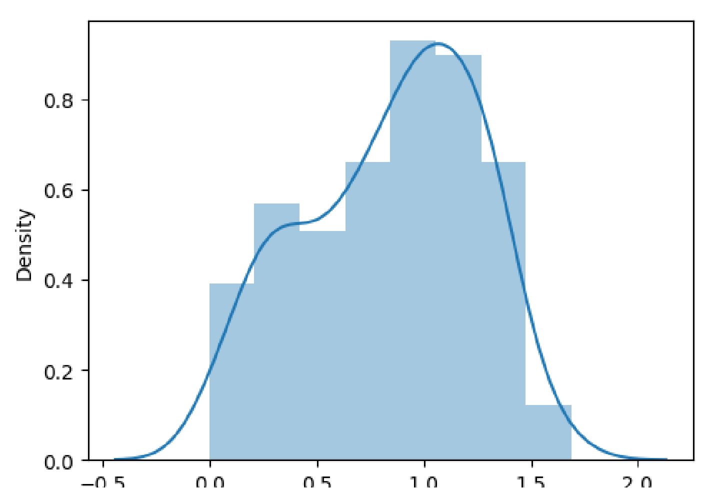
Chart - 13

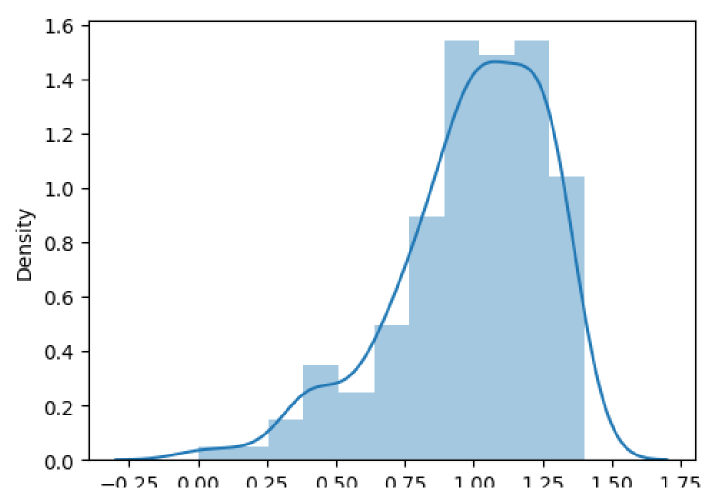
Distribution Plot

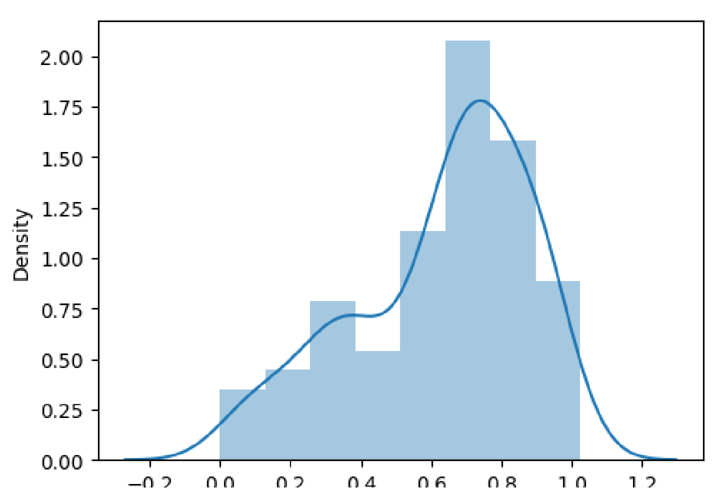
```
nums_columns = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

plt.figure(figsize=(20,15))
plotnumber=1

for column in nums_columns:
  if plotnumber<=16:
    ax=plt.subplot(4,4,plotnumber)
    sns.distplot(nums_columns[column])
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.tight_layout()

plt.show()
```

```
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
<ipython-input-23-c75fe795adc6>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(nums_columns[column])
```









Insights from above chart:

1. Standard Error: Positively skewed with most countries having low standard errors, but a few outliers with significantly higher values.
2. Economy (GDP per Capita): Approximately normally distributed, suggesting that GDP per capita varies but is roughly normally distributed.
3. Family: Somewhat right-skewed, indicating variation in social support within families across countries.
4. Health (Life Expectancy): Approximately normally distributed with most countries having high life expectancy.
5. Freedom: Somewhat right-skewed, with varying levels of perceived freedom across countries.
6. Trust (Government Corruption): Right-skewed, with most countries having low trust in governments and a few with higher trust levels.
7. Generosity: Approximately normally distributed, showing variation in generosity levels.
8. Dystopia Residual: Right-skewed, with most countries having lower values, but some with higher values.

Chart - 14

Pair Plot

```
sns.pairplot(happiness_data)

plt.show()
```

Will Do encoding for our variables, for implementing ML Model.

```
from sklearn.preprocessing import LabelEncoder


# Create LabelEncoder instances for 'Country' and 'Region'
country_encoder = LabelEncoder()
region_encoder = LabelEncoder()

# Fit and transform the 'Country' and 'Region' columns
happiness_data['Country'] = country_encoder.fit_transform(happiness_data['Country'])
happiness_data['Region'] = region_encoder.fit_transform(happiness_data['Region'])
```

ML Model Implementation

ML Model - 1

Using all varibales

```
# Importing Necessary Libraries

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import Ridge
import math
```

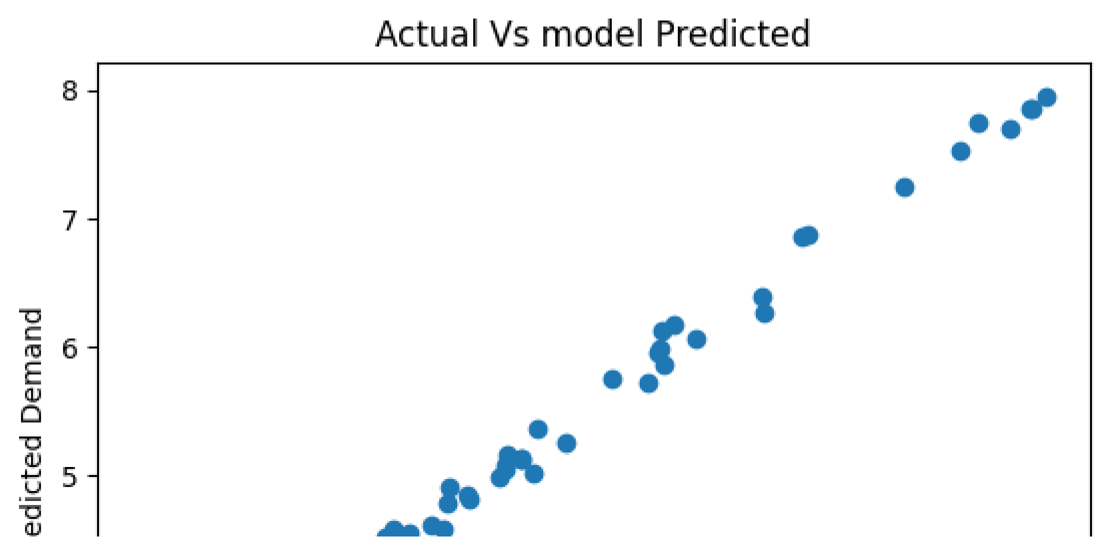
```
happiness_data.columns
```

```
     Index(['Country', 'Region', 'Happiness_Rank', 'Happiness_Score',
            'Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
            'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
            'Generosity', 'Dystopia_Residual'],
           dtype='object')
```

```
x = happiness_data[['Country', 'Region', 'Happiness_Rank',
       'Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 1: ", LR_mse)
print("Linear Regression RMSE For Model 1: ", LR_RMSE)
print("Linear Regression R-squared For Model 1: ", LR_r2)
```

```
plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()
```

```
Shape of x_train (110, 11)
Shape of x_test (48, 11)
Shape of y_train (110,)
Shape of y_train (48,)
Linear Regression MSE For Model 1:  0.07497957560237652
Linear Regression RMSE For Model 1:  0.27382398653583384
Linear Regression R-squared For Model 1:  0.9354051123234662
```



Insights from ML Model 1:

1. Based on these evaluation metrics, the linear regression model appears to be performing very well. The low MSE and RMSE values, along with the high R2 score, indicate that the model provides accurate and reliable predictions of happiness scores.But Seems Like overfiting model, will check for other models as well.

Will Check for lasso and ridge regression on same.

```
# Importing Necessary Libraries

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE: ", lasso_mse)
print("Lasso Regression R-squared: ", lasso_r2)
print("Best Lasso Alpha: ", lasso_grid.best_params_['alpha'])


# Similarly for ridge regression


# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE: ", ridge_mse)
print("Ridge Regression R-squared: ", ridge_r2)
print("Best Ridge Alpha: ", ridge_grid.best_params_['alpha'])
```

```
Lasso Regression MSE:  0.023618971650834385
Lasso Regression R-squared:  0.9796522611849443
Best Lasso Alpha:  0.01
Ridge Regression MSE:  0.06595545710249974
Ridge Regression R-squared:  0.943179388400601
Best Ridge Alpha:  0.01
```

Both Lasso and Ridge regression models seem to perform well, with Lasso having slightly lower MSE and a slightly higher R2 value. The high R2 values for both models indicate that they are doing a good job of explaining the variance in the "Happiness_Score" based on the input features.

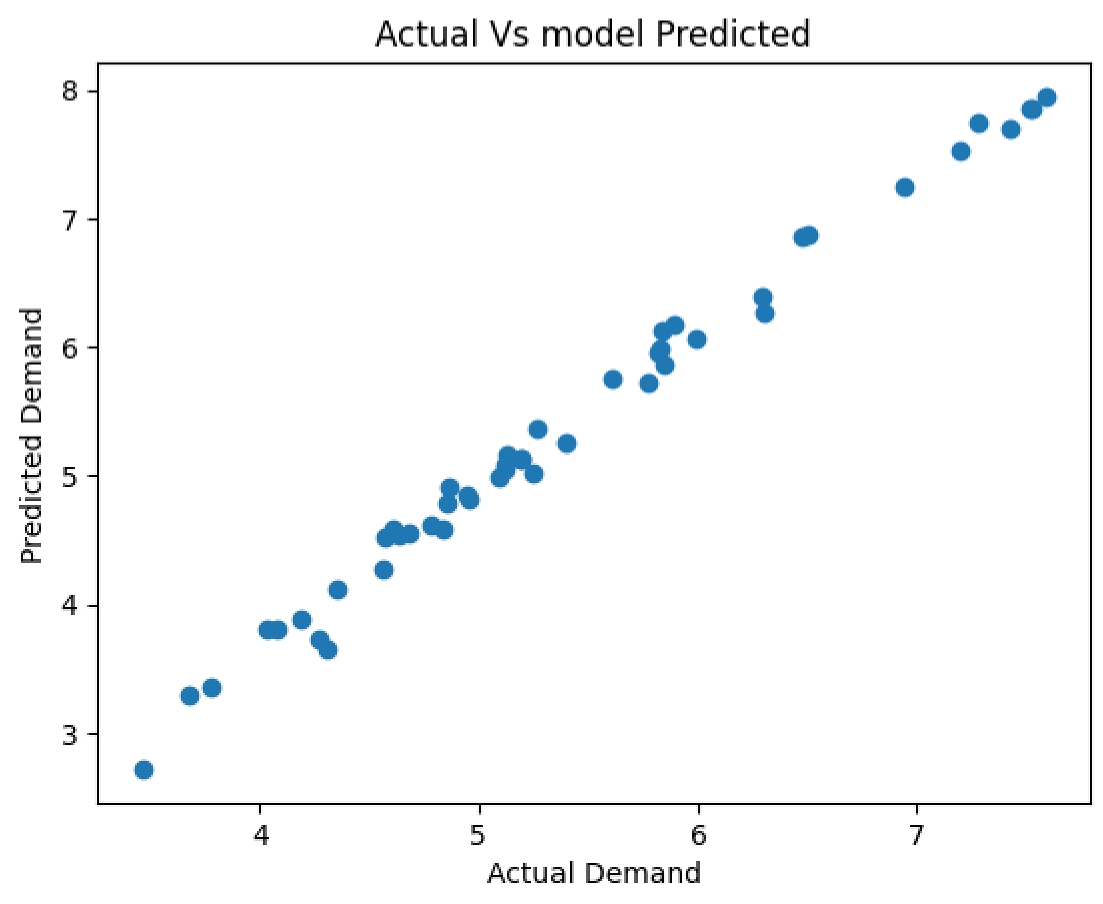
ML Model - 2

Using numerical variables

```
x = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)
```

```
# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 2: ", LR_mse)
print("Linear Regression RMSE For Model 2: ", LR_RMSE)
print("Linear Regression R-squared For Model 2: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()


# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 2: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 2: ", lasso_r2)
print("Best Lasso Alpha For ML Model 2: ", lasso_grid.best_params_['alpha'])


# Similarly for ridge regression


# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 2: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 2: ", ridge_r2)
print("Best Ridge Alpha For ML Model 2: ", ridge_grid.best_params_['alpha'])
```

```
Shape of x_train (110, 8)
Shape of x_test (48, 8)
Shape of y_train (110,)
Shape of y_train (48,)
Linear Regression MSE For Model 2:  0.07499194441377484
Linear Regression RMSE For Model 2:  0.273846570936674
Linear Regression R-squared For Model 2:  0.9353944565952023
```



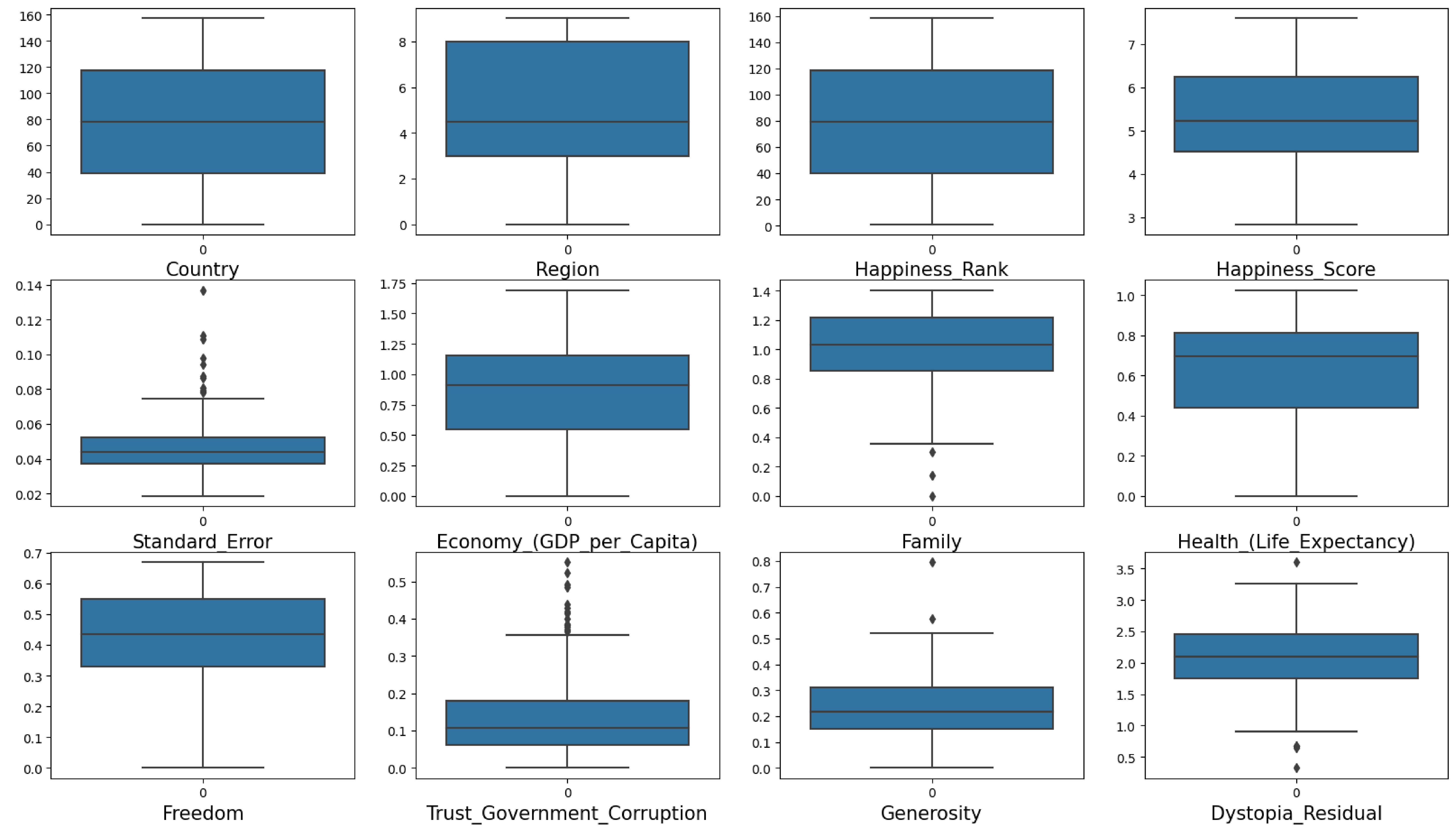
```
Lasso Regression MSE for ML Model 2:  0.03927181284270662
Lasso Regression R-squared for ML Model 2:  0.9661673419854875
Best Lasso Alpha For ML Model 2:  0.01
Ridge Regression MSE For ML Model 2:  0.07379956198065292
Ridge Regression R-squared For ML Model 2:  0.9364216937956825
Best Ridge Alpha For ML Model 2:  0.01
```

Insights from ML Model 2:

1. The Linear Regression model (Model 2) has a relatively low MSE and RMSE, indicating that it provides a reasonably good fit to the data. The smaller the MSE and RMSE, the better the model's predictive accuracy.
2. The R-squared (R2) value of 0.935 suggests that approximately 93.5% of the variance in the Happiness_Score can be explained by the independent variables in the model. This is a strong R2 value, signifying that the model captures a substantial portion of the variability in the target variable.
3. Lasso Regression performs even better than Model 2 in terms of MSE and R-squared. It has a significantly lower MSE, indicating improved predictive accuracy, and an impressive R2 value of 0.966, suggesting that it explains a large portion of the variance in Happiness_Score.
4. Ridge Regression also shows strong performance with a low MSE and a high R-squared value. The MSE is slightly higher than that of the Lasso model but still indicates a good fit. The R2 value of 0.936 suggests that it explains a large part of the variance in Happiness_Score.

Now will check for outliers in dataset.

```
num_columns = happiness_data[['Country', 'Region', 'Happiness_Rank',
       'Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

plt.figure(figsize=(20,15))
graph = 1

for column in happiness_data:
  if graph<=16:
    plt.subplot(4,4,graph)
    ax=sns.boxplot(data= happiness_data[column])
    plt.xlabel(column,fontsize=15)
  graph+=1
plt.show()
```



Insights from above plot:

1. From above box plo we can observe that we have very outliers iin very few variables like standard Error, Family, Trust_(Government_Corruption), Generosity, Dystopia_Residual. Will check with z-threshold limit for treatment on outliers.

```
# Will check for quintile data

q1 = happiness_data.quantile(0.25)

q3 = happiness_data.quantile(0.75)

q2 = happiness_data.quantile(0.50)

iqr = q3 - q1


# Using outlier detection formula

# For Higher Side

Standard_Error_high = (q3.Standard_Error + (1.5 * iqr.Standard_Error)) # Standard_Error higher

Standard_Error_high

# Check the indexes which have higher values

np_index = np.where(happiness_data['Standard_Error'] > Standard_Error_high)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape


# For Family have outliers in lower side so,

Family_lower = (q1.Family - (1.5 * iqr.Family)) # Family lower

Family_lower

# Check the indexes which have higher values
```

```
np_index = np.where(happiness_data['Family'] < Family_lower)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape
```

```
(145, 12)
```

```
happiness_data.columns
```

```
Index(['Country', 'Region', 'Happiness_Rank', 'Happiness_Score',
       'Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual'],
      dtype='object')
```

```
# In trust variable having outliers in Higher Side

Trust_Government_Corruption_high = (q3.Trust_Government_Corruption + (1.5 * iqr.Trust_Government_Corruption)) # Trust_(Government_Corruption) higher

Trust_Government_Corruption_high

# Check the indexes which have higher values

np_index = np.where(happiness_data['Trust_Government_Corruption'] > Trust_Government_Corruption_high)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape

# In Genoricity variable having outliers in Higher Side

Generosity_high = (q3.Generosity + (1.5 * iqr.Generosity)) # Trust_(Government_Corruption) higher

Generosity_high

# Check the indexes which have higher values

np_index = np.where(happiness_data['Generosity'] > Generosity_high)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape
```

```
(129, 12)
```

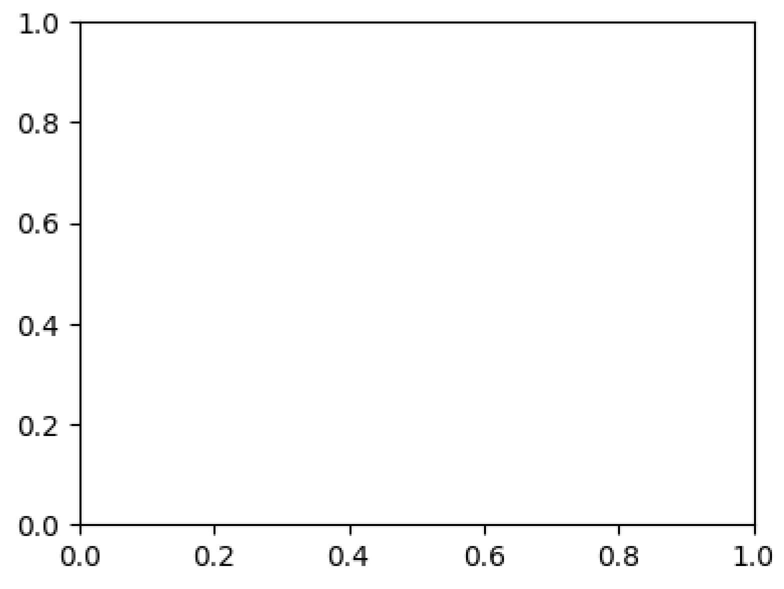
```
# In Genoricity variable having outliers in Higher Side and lower side

Dystopia_Residual_high = (q3.Dystopia_Residual + (1.5 * iqr.Dystopia_Residual)) # Trust_(Government_Corruption) higher

Dystopia_Residual_high

# Check the indexes which have higher values

np_index = np.where(happiness_data['Dystopia_Residual'] > Dystopia_Residual_high)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape

# For lower side,

Dystopia_Residual_lower = (q1.Dystopia_Residual - (1.5 * iqr.Dystopia_Residual)) # Family lower

Dystopia_Residual_lower

# Check the indexes which have higher values

np_index = np.where(happiness_data['Dystopia_Residual'] < Dystopia_Residual_lower)
np_index

# Drop the index which we found in the above cell

data = happiness_data.drop(happiness_data.index[np_index],inplace=True)

happiness_data.shape
```

```
(126, 12)
```

```
num_columns = happiness_data[['Country', 'Region', 'Happiness_Rank',
       'Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

plt.figure(figsize=(20,15))
graph = 1

for column in happiness_data:
  if graph<=16:
    plt.subplot(4,4,graph)
    ax=sns.boxplot(data= happiness_data[column])
    plt.xlabel(column,fontsize=15)
  graph+=1
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

8 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 0
```

SEARCH STACK OVERFLOW

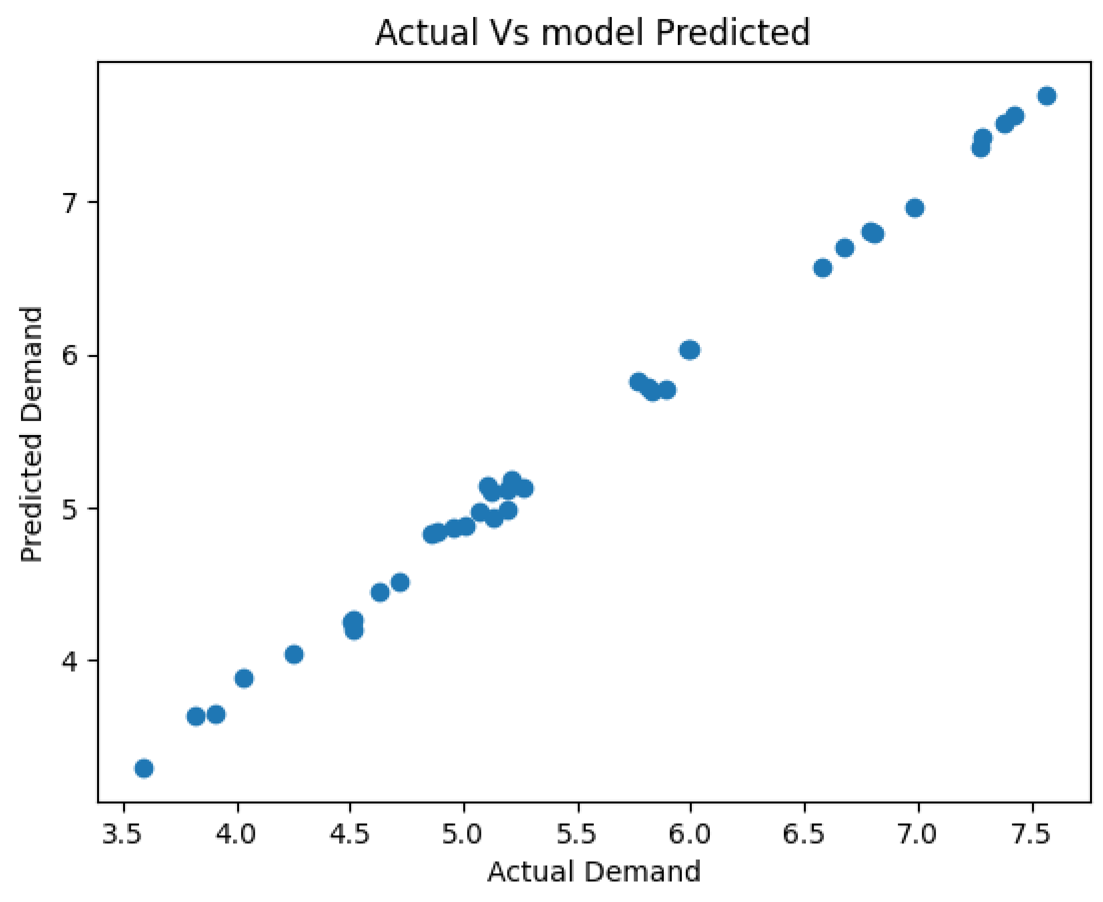


ML Model - 3

ML Model after Treatment of outliers

```
x = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 3: ", LR_mse)
print("Linear Regression RMSE For Model 3: ", LR_RMSE)
print("Linear Regression R-squared For Model 3: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()


# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 3: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 3: ", lasso_r2)
print("Best Lasso Alpha For ML Model 3: ", lasso_grid.best_params_['alpha'])


# Similarly for ridge regression


# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 3: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 3: ", ridge_r2)
print("Best Ridge Alpha For ML Model 3: ", ridge_grid.best_params_['alpha'])
```

```
Shape of x_train (88, 8)
Shape of x_test (38, 8)
Shape of y_train (88,)
Shape of y_train (38,)
Linear Regression MSE For Model 3:  0.020581671132573944
Linear Regression RMSE For Model 3:  0.143463135099488
Linear Regression R-squared For Model 3:  0.9830474185633036
```



```
Lasso Regression MSE for ML Model 3:  0.018093864343124405
Lasso Regression R-squared for ML Model 3:  0.985096559613379
Best Lasso Alpha For ML Model 3:  0.01
Ridge Regression MSE For ML Model 3:  0.02028061142695857
Ridge Regression R-squared For ML Model 3:  0.9832953935282069
Best Ridge Alpha For ML Model 3:  0.01
```

Insights from ML Model 3:

1. The Linear Regression model (Model 3) performs quite well. It has a very low MSE and RMSE, indicating that it provides an excellent fit to the data. The smaller the MSE and RMSE, the better the model's predictive accuracy.
2. The R-squared (R2) value of 0.983 suggests that approximately 98.3% of the variance in the Happiness_Score can be explained by the independent variables in the model. This is a very strong R2 value
3. Lasso Regression performs even better than Model 3 in terms of MSE and R-squared. It has a significantly lower MSE, indicating improved predictive accuracy, and an impressive R2 value of 0.985, suggesting that it explains a large portion of the variance in Happiness_Score.
4. Ridge Regression also shows strong performance with a low MSE and a high R-squared value. The MSE is slightly higher than that of Lasso but still indicates a good fit. The R2 value of 0.983 suggests that it explains a large part of the variance in Happiness_Score.

ML Model - 4

knn Regression Model

```
# Imporing Library

from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler



x = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# Normalization
scalar = StandardScaler()

x_scalar = scalar.fit_transform(happiness_data)

best_features = SelectKBest(score_func=f_classif,k=1)

fit = best_features.fit(x,y)

df_score = pd.DataFrame(fit.scores_)

df_columns = pd.DataFrame(x.columns)


feature_scores = pd.concat([df_columns,df_score],axis=1)

feature_scores.columns = ['Feature_Name','Score']
```

```
print(feature_scores.nlargest(1,'Score'))

# Fitting knn Model
knn = KNeighborsRegressor(n_neighbors=5)

knn.fit(x_train, y_train)

# Evaluting matrix
def metric_score(model, x_train, x_test, y_train, y_test, train=True):
    if train:
        y_pred = model.predict(x_train)
    else:
        y_pred = model.predict(x_test)

    mse = mean_squared_error(y_train if train else y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_train if train else y_test, y_pred)

    return mse, rmse, r2

# Calculate training and testing scores
train_mse, train_rmse, train_r2 = metric_score(knn, x_train, x_test, y_train, y_test, train=True)
test_mse, test_rmse, test_r2 = metric_score(knn, x_train, x_test, y_train, y_test, train=False)

print("Training MSE:", train_mse)
print("Training RMSE:", train_rmse)
print("Training R2:", train_r2)

print("Testing MSE:", test_mse)
print("Testing RMSE:", test_rmse)
print("Testing R2:", test_r2)

from sklearn.model_selection import KFold,cross_val_score
k_f =KFold(n_splits=5)

k_f

for train,test in k_f.split([12,23,35,46,51,63,75,86,96,108]):
  print('train : ',train,'test :',test)

cross_val_score(knn,x_scalar,y,cv=5)
cross_val_score(knn,x_scalar,y,cv=5).mean

from sklearn.model_selection import GridSearchCV
param_grid = {'algorithm': ['kd_tree','brute'],
              'leaf_size':[3,5,6,7,8],
              'n_neighbors': [3,5,7,9,11,13]}

gridsearch = GridSearchCV(estimator =knn,param_grid=param_grid)

gridsearch.fit(x_train,y_train)

gridsearch.best_score_

gridsearch.best_estimator_

metric_score(knn,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(knn,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
                 Feature_Name         Score
5  Trust_Government_Corruption  1.536284e+07
Training MSE: 0.07036308500000005
Training RMSE: 0.265260409786308
Training R2: 0.926120785423518
Testing MSE: 0.1385488252631578
Testing RMSE: 0.37222147340415196
Testing R2: 0.8858809749653908
train :  [2 3 4 5 6 7 8 9] test : [0 1]
train :  [0 1 4 5 6 7 8 9] test : [2 3]
train :  [0 1 2 3 6 7 8 9] test : [4 5]
train :  [0 1 2 3 4 5 8 9] test : [6 7]
train :  [0 1 2 3 4 5 6 7] test : [8 9]
(0.1385488252631578, 0.37222147340415196, 0.8858809749653908)
```

Insights from kNN Regression Model

1. The training MSE of 0.0704 is relatively low, suggesting that the KNN model fits the training data quite well. Lower MSE indicates better model performance.The training R2 value of 0.9261 indicates that the KNN model explains approximately 92.6% of the variance in the target variable (Happiness_Score) within the training dataset.
2. The testing MSE of 0.1385 is higher than the training MSE but still relatively reasonable. It indicates that the model's predictive accuracy is good on the testing data, although not as good as on the training data.The testing R2 value of 0.8859 indicates that the KNN model explains approximately 88.6% of the variance in the "Happiness_Score" within the testing dataset.
3. The KNN model appears to be quite effective for predicting "Happiness_Score." It performs well on both the training and testing datasets, with low MSE and RMSE values and high R-squared values.

ML Model - 5

Decision Tree Regression Model

```
from sklearn.tree import DecisionTreeRegressor

x = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# Train and test set split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=348)

# Fitting Model
clf = DecisionTreeRegressor()
```

```
clf.fit(x_train,y_train)

# defining function for evalution matrix
def metric_score(model, x_train, x_test, y_train, y_test, train=True):
    if train:
        y_pred = model.predict(x_train)
        mse = mean_squared_error(y_train, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_train, y_pred)
        print('\n======Train Result======')
        print(f'Mean Squared Error (MSE): {mse:.2f}')
        print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
        print(f'R-squared (R2): {r2:.2f}')
    else:
        y_pred = model.predict(x_test)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)
        print('\n======Test Result======')
        print(f'Mean Squared Error (MSE): {mse:.2f}')
        print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
        print(f'R-squared (R2): {r2:.2f}')

# Calling above function and passing dataset to check train and test score

metric_score(clf,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(clf,x_train,x_test,y_train,y_test,train=False)  # for testing score

# Now doing Hypertuning

grid_param = {
    'criterion': ['squared_error'],
    'max_depth': range(5, 10),
    'min_samples_leaf': range(1, 3),
    'min_samples_split': range(1, 5),
    'max_leaf_nodes': range(3, 6)
}

grid_search = GridSearchCV(estimator=clf,
                           param_grid=grid_param,
                           cv=5,
                           n_jobs=-1,
                           error_score=np.nan)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print(best_parameters)

# Using best_param for model

clf = DecisionTreeRegressor(criterion='squared_error',min_samples_split=3,max_depth=5,min_samples_leaf=1)

clf.fit(x_train,y_train)

metric_score(clf,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(clf,x_train,x_test,y_train,y_test,train=False)  # for testing score
```

```
Mean Squared Error (MSE): 0.22
Root Mean Squared Error (RMSE): 0.47
R-squared (R2): 0.68
{'criterion': 'squared_error', 'max_depth': 7, 'max_leaf_nodes': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}

======Train Result======
Mean Squared Error (MSE): 0.03
Root Mean Squared Error (RMSE): 0.17
R-squared (R2): 0.97

======Test Result======
Mean Squared Error (MSE): 0.22
Root Mean Squared Error (RMSE): 0.46
R-squared (R2): 0.68
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
150 fits failed out of a total of 600.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
150 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py", line 1247, in fit
    super().fit(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py", line 177, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_samples_split' parameter of DecisionTreeRegressor must be an int in the range [2, inf) or a float in the range (0.0, 1.0]

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite: [       nan 0.50294944 0.50294944 0.50294944
 0.50294944 0.50294944        nan 0.46926578 0.46926578 0.46926578
        nan 0.46926578 0.46926578 0.46926578        nan 0.56089999
 0.56089999 0.56089999        nan 0.56089999 0.56089999 0.56089999
        nan 0.50294944 0.50294944 0.50294944        nan 0.50294944
 0.50294944 0.50294944        nan 0.46926578 0.46926578 0.46926578
        nan 0.46926578 0.46926578 0.46926578        nan 0.56089999
 0.56089999 0.56089999        nan 0.56089999 0.56089999 0.56089999
        nan 0.50294944 0.50294944 0.50294944        nan 0.50294944
 0.50294944 0.50294944        nan 0.46926578 0.46926578 0.46926578
        nan 0.46926578 0.46926578 0.46926578        nan 0.56089999
 0.56089999 0.56089999        nan 0.56089999 0.56089999 0.56089999
```

```
  0.98089999 0.98089999        nan 0.98089999 0.98089999 0.98089999]
  warnings.warn(
```

Insights from Decision Tree Regression Model

1. The training results suggest a perfect fit (R2 = 1.00), indicating that the model fits the training data perfectly without errors. However, the test results show a significant drop in performance, with an MSE of 0.22 and an R2 of 0.68. This indicates that the model did not generalize well to unseen data and might be overfitting.
2. Hyperparameters used in the tuned model: {'criterion': 'squared_error', 'max_depth': 7, 'max_leaf_nodes': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}. After hyperparameter tuning, the training and test results have improved. The training R2 is 0.97, which suggests a strong fit to the training data.
3. The test results (MSE and R2) remain the same as in the original model. This indicates that while hyperparameter tuning improved the model's fit to the training data, it did not significantly enhance its generalization to the test data.

ML Model - 5

RandomForesteRegressor

```
from sklearn.ensemble import RandomForestRegressor

x = happiness_data[['Standard_Error', 'Economy_(GDP_per_Capita)', 'Family',
       'Health_(Life_Expectancy)', 'Freedom', 'Trust_Government_Corruption',
       'Generosity', 'Dystopia_Residual']]

y = happiness_data['Happiness_Score']

# Train and test set split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=348)

# Random Forest Regression
rf_model = RandomForestRegressor()

rf_model.fit(x_train,y_train)

# defining function for evalution matrix
def metric_score(model, x_train, x_test, y_train, y_test, train=True):
    if train:
        y_pred = model.predict(x_train)
        mse = mean_squared_error(y_train, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_train, y_pred)
        print('\n======Train Result======')
        print(f'Mean Squared Error (MSE): {mse:.2f}')
        print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
        print(f'R-squared (R2): {r2:.2f}')
    else:
        y_pred = model.predict(x_test)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)
        print('\n======Test Result======')
        print(f'Mean Squared Error (MSE): {mse:.2f}')
        print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
        print(f'R-squared (R2): {r2:.2f}')

# Calling above function and passing dataset to check train and test score

metric_score(rf_model,x_train,x_test,y_train,y_test,train=True) # for training score

metric_score(rf_model,x_train,x_test,y_train,y_test,train=False)  # for testing score


# Now doing Hypertuning

rf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}

grid_search = GridSearchCV(estimator=rf_model,
                           param_grid=rf_params,
                           cv=5,
                           n_jobs=-1,
                           scoring='neg_mean_squared_error')

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print(best_parameters)

# Using the best parameters for the model
best_rf_model = RandomForestRegressor(**best_parameters)
best_rf_model.fit(x_train, y_train)

# Check scores for the model with the best parameters
metric_score(best_rf_model, x_train, x_test, y_train, y_test, train=True)  # for training score
metric_score(best_rf_model, x_train, x_test, y_train, y_test, train=False)  # for testing score
```

```
======Train Result======
Mean Squared Error (MSE): 0.02
Root Mean Squared Error (RMSE): 0.13
R-squared (R2): 0.99

======Test Result======
Mean Squared Error (MSE): 0.08
Root Mean Squared Error (RMSE): 0.28
R-squared (R2): 0.88
{'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 300}

======Train Result======
Mean Squared Error (MSE): 0.01
Root Mean Squared Error (RMSE): 0.12
R-squared (R2): 0.99
```