World Happiness Report Project

```
* Internship Practice Project Phase-2 at FlipRobo Technologies
* Author Name: AYAZ WARIS KHAN
* Batch No: DS2307
```

1. Import Libraries / Modules

In [1]:
```
#Importing Libraries
import numpy as np
import pandas as pd

# To Visualize the Data
import seaborn as sns
import matplotlib.pyplot as plt

# To prevent WARNINGS!
import warnings
warnings.filterwarnings('ignore')
```

2. Import and Analyze the Data

In [2]:
```
#Loading Dataset
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/ha
df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.5
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.4
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.1
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	0.3

158 rows × 12 columns

In [3]:
```
#checking the shape of dataset
print("There are {} rows and {} columns respectively present in the dataset.".f
```

```
There are 158 rows and 12 columns respectively present in the dataset.
```

In [4]:
```python
#Columns present in our dataset
df.columns
```

Out[4]:
```
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 [5]:
```python
# Detail of columns
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
```

Observation:

- The dataset comprises 12 columns with three distinct data types.
- Nine columns contain floating-point values, one column contains integer values, and two columns contain object values.
- The memory usage for this dataset is approximately 14.9 KB.

Based on the provided information, there are no apparent null values in the dataset. Now, let's explore each column in more detail:

1. **Country:** This column represents the names of different countries.
2. **Region:** The "Region" column indicates the region to which each country belongs.
3. **Happiness Rank:** This column signifies the ranking of countries based on their Happiness Score.
4. **Happiness Score:** It is a metric measured in 2015 by asking sampled individuals the question: "On a scale of 0 to 10, where 10 represents the highest level of happiness, how would you rate your happiness?"
5. **Standard Error:** This column represents the standard error associated with the Happiness Score, providing information about the precision of the happiness score measurement for

each country.

6. **Economy (GDP per Capita):** This column quantifies the extent to which a country's GDP per capita contributes to its overall Happiness Score, reflecting economic well-being.
7. **Family:** It measures the contribution of family and social support to the Happiness Score, indicating the strength of social relationships within a country.
8. **Health (Life Expectancy):** This column represents the contribution of life expectancy to the Happiness Score, reflecting the health and well-being of the population.
9. **Freedom:** It quantifies the extent to which freedom contributes to the Happiness Score, measuring personal and political freedoms within a country.
10. **Trust (Government Corruption):** This column reflects the contribution of trust in government and the absence of corruption to the Happiness Score, where lower corruption and higher trust can lead to greater happiness.
11. **Generosity:** It measures the extent to which generosity among the population contributes to the Happiness Score, reflecting the willingness of individuals to help others.
12. **Dystopia Residual:** This column represents a hypothetical country with the lowest possible Happiness Score. It serves as a reference point for comparing and calculating the impact of all other factors on happiness.

In [6]:
```
#Renaming the Columns for our convenience
df.rename(columns={
    "Happiness Rank": "Happiness_Rank",
    "Happiness Score": "Happiness_Score",
    "Standard Error": "Standard_Error",
    "Economy (GDP per Capita)": "Economy",
    "Health (Life Expectancy)": "Health_Life_Expectancy",
    "Trust (Government Corruption)": "Trust_Government_Corruption",
    "Dystopia Residual": "Dystopia_Residual"
}, inplace=True)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['Country', 'Region', 'Happiness_Rank', 'Happiness_Score',
       'Standard_Error', 'Economy', 'Family', 'Health_Life_Expectancy',
       'Freedom', 'Trust_Government_Corruption', 'Generosity',
       'Dystopia_Residual'],
      dtype='object')
```

Observations:

- All columns have been renamed to use simpler names.

In [8]:
```
# checking unique values in our dataframe
df.nunique()
```

Out[8]:
```
Country                          158
Region                            10
Happiness_Rank                   157
Happiness_Score                  157
Standard_Error                   153
Economy                          158
Family                           158
Health_Life_Expectancy           157
Freedom                          158
Trust_Government_Corruption      157
Generosity                       158
Dystopia_Residual                158
dtype: int64
```

Observations:

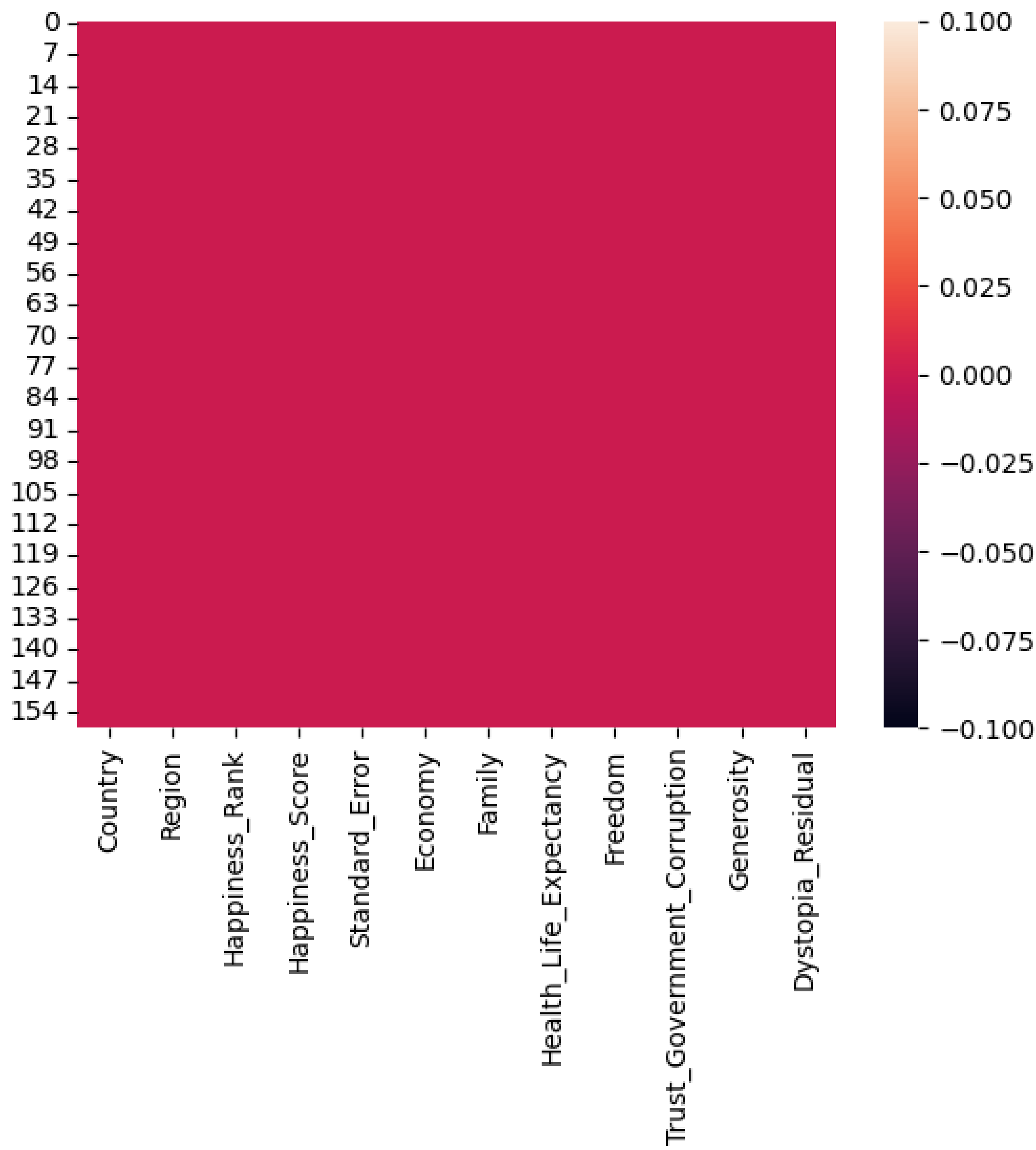
It's evident that each column contains over 150 unique values, except for the "Region" column, which has only 10 distinct values.

In [9]:
```
#Checking for missing values in our data
df.isnull().sum()
```

Out[9]:
```
Country                          0
Region                           0
Happiness_Rank                   0
Happiness_Score                  0
Standard_Error                   0
Economy                          0
Family                           0
Health_Life_Expectancy           0
Freedom                          0
Trust_Government_Corruption      0
Generosity                       0
Dystopia_Residual                0
dtype: int64
```

In [10]:
```
#lets visualise it
sns.heatmap(df.isnull())
```

Out[10]: <Axes: >



Observations:

- We can clearly see that there are no null values in our dataframe.

In [11]:
```
#checking for duplicate values
print("There are {} duplicates present in the dataset".format(df.duplicated().s
```

```
There are 0 duplicates present in the dataset
```

Separating Numerical and Categorical Columns

In [12]:
```python
numerical_columns = df.select_dtypes(include=['int', 'float']).columns.tolist()
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

print("Numerical Columns:\n",numerical_columns)
print("\nCategorical Columns:\n",categorical_columns)
```

```
Numerical Columns:
 ['Happiness_Rank', 'Happiness_Score', 'Standard_Error', 'Economy', 'Family',
'Health_Life_Expectancy', 'Freedom', 'Trust_Government_Corruption', 'Generosi
ty', 'Dystopia_Residual']

Categorical Columns:
 ['Country', 'Region']
```

Observations: These are numerical columns & Categorical columns in our dataframe

In [13]:
```python
#statistical Summary of numerical columns
df.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	
Happiness_Rank	158.0	79.493671	45.754363	1.00000	40.250000	79.500000	118.75
Happiness_Score	158.0	5.375734	1.145010	2.83900	4.526000	5.232500	6.24
Standard_Error	158.0	0.047885	0.017146	0.01848	0.037268	0.043940	0.05
Economy	158.0	0.846137	0.403121	0.00000	0.545808	0.910245	1.15
Family	158.0	0.991046	0.272369	0.00000	0.856823	1.029510	1.21
Health_Life_Expectancy	158.0	0.630259	0.247078	0.00000	0.439185	0.696705	0.81
Freedom	158.0	0.428615	0.150693	0.00000	0.328330	0.435515	0.54
Trust_Government_Corruption	158.0	0.143422	0.120034	0.00000	0.061675	0.107220	0.18
Generosity	158.0	0.237296	0.126685	0.00000	0.150553	0.216130	0.30
Dystopia_Residual	158.0	2.098977	0.553550	0.32858	1.759410	2.095415	2.46

Observations: This dataset comprises 158 observations, each corresponding to a different country. It provides insights into various factors influencing a country's happiness and well-being.

- **Happiness Rank:** Ranging from 1 to 158, this column indicates a country's relative position in terms of happiness compared to others.
- **Happiness Score:** With values between 2.839 and 7.587 and an average score of 5.375734, this metric quantifies the overall happiness level of a country, considering multiple contributing factors.
- **Standard Error:** This column reflects the standard error of the happiness score estimation for each country, with an average value of 0.047885.
- **Economy (GDP per Capita):** Ranging from 0 to 1.69042, with an average of 0.846137, this metric provides insight into a country's GDP per capita, an important factor in happiness assessment.
- **Family:** With values from 0 to 1.40223 and an average of 0.991046, this column represents the strength of social support and relationships within a country.

- **Health (Life Expectancy):** Ranging from 0 to 1.02525, with an average of 0.630259, this metric indicates the life expectancy of a country's population, a key determinant of overall happiness.
- **Freedom:** Ranging from 0 to 0.66973, with an average of 0.428615, this metric measures the perceived level of freedom and autonomy within a country.
- **Trust (Government Corruption):** With values between 0 and 0.55191 and an average of 0.143422, this column reflects the perception of government corruption within each country.
- **Generosity:** Ranging from 0 to 0.79588, with an average of 0.237296, this metric indicates the generosity and charitable behavior of individuals within a country.
- **Dystopia Residual:** With values from 0.32858 to 3.60214 and an average of 2.098977, this metric quantifies the extent to which the happiness score is influenced by unaccounted factors not included in the dataset.

3. Exploratory Data Analysis

Univariate Analysis

In [14]:
```
#Exploring Our Target Column Happiness Score.
df['Happiness_Score'].value_counts().to_frame("Unique Values")
```

Out[14]:

	Unique Values
5.192	2
7.587	1
4.686	1
4.839	1
4.800	1
...	...
5.855	1
5.848	1
5.833	1
5.828	1
2.839	1

157 rows × 1 columns

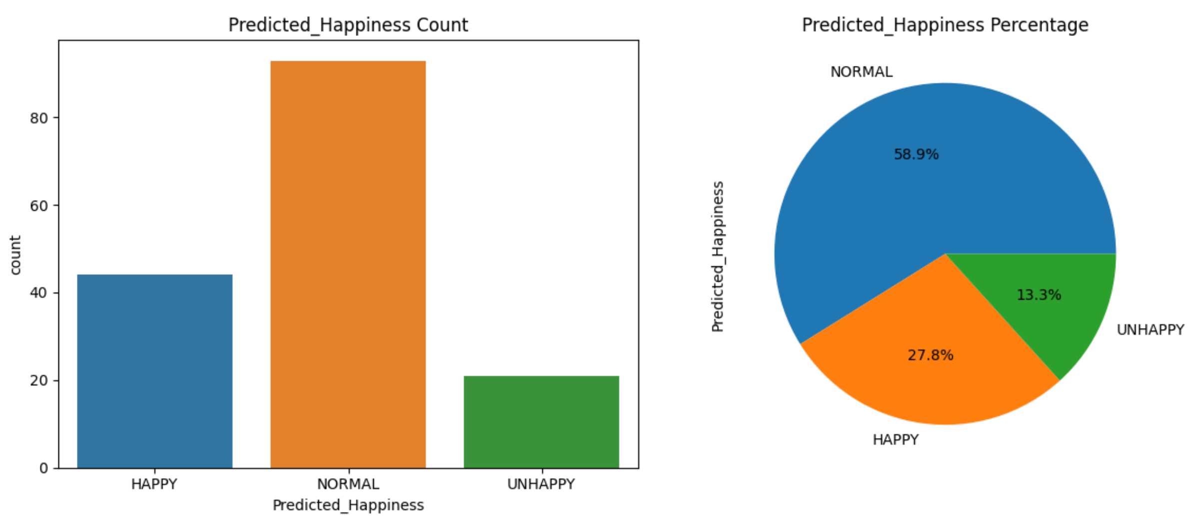
Observations:

- As per the domain Knowledge and clear observation we can conclude that "happiness score" column contains values in range of 2.83900 to 7.587. So we can create a new column based on the Happiness Score

In [15]:
```python
# Create a list to store the predicted happiness category
happiness = []

for score in df['Happiness_Score']:
    if score < 4:
        happiness.append("UNHAPPY")
    elif score >= 4 and score <= 6:
        happiness.append("NORMAL")
    else:
        happiness.append("HAPPY")

pred_happiness = pd.DataFrame(happiness, columns=["Predicted_Happiness"])
pred_happiness = pred_happiness.astype('category')
data = pd.concat([df, pred_happiness], axis=1)
```

In [16]:
```python
#Checkig the Distribution of Hapiness Score
print(data['Predicted_Happiness'].value_counts())

# Checking the Predicted Happiness
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.countplot(x='Predicted_Happiness', data=data, ax=axes[0])
axes[0].set_title("Predicted_Happiness Count")

# Checking the Survived percentage
data['Predicted_Happiness'].value_counts().plot(kind='pie', autopct='%0.1f%%',
axes[1].set_title("Predicted_Happiness Percentage")
plt.tight_layout()
plt.show()
```

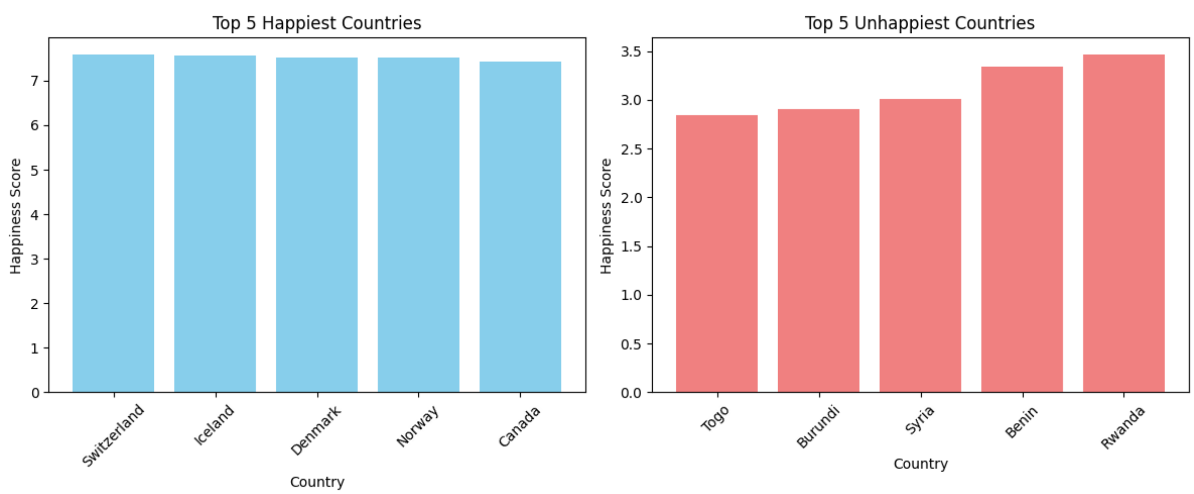
```
NORMAL     93
HAPPY      44
UNHAPPY    21
Name: Predicted_Happiness, dtype: int64
```



Observations:

- We can clearly visualise by value count & percentage that countires with Normal Happiness

In [17]:
```
#Checking Top 5 Happy & Unhappy countries

# Sort the data by 'Happiness_Score' in descending order and select the top 5 h
top_5_happiest_countries = data.sort_values('Happiness_Score', ascending=False)
top_5_unhappiest_countries = data.sort_values('Happiness_Score', ascending=True

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Loop to create both bar charts
for i, (countries, title, color) in enumerate(
    [(top_5_happiest_countries, 'Top 5 Happiest Countries', 'skyblue'),
     (top_5_unhappiest_countries, 'Top 5 Unhappiest Countries', 'lightcoral')])
    ax = axes[i]
    ax.bar(countries['Country'], countries['Happiness_Score'], color=color)
    ax.set_xlabel('Country')
    ax.set_ylabel('Happiness Score')
    ax.set_title(title)
    ax.tick_params(axis='x', rotation=45)

# Adjust spacing between subplots
plt.tight_layout()
# Show the combined figure
plt.show()
```



Observations:

- From the above plot, we can clearly visualize that Switzerland is the most happiest Country & Togo is the most unhappy country.

Dropping columns

In [18]:
```
data = data.drop(['Country', 'Region', 'Happiness_Rank','Predicted_Happiness'],
data.head(5)
```

Out[18]:

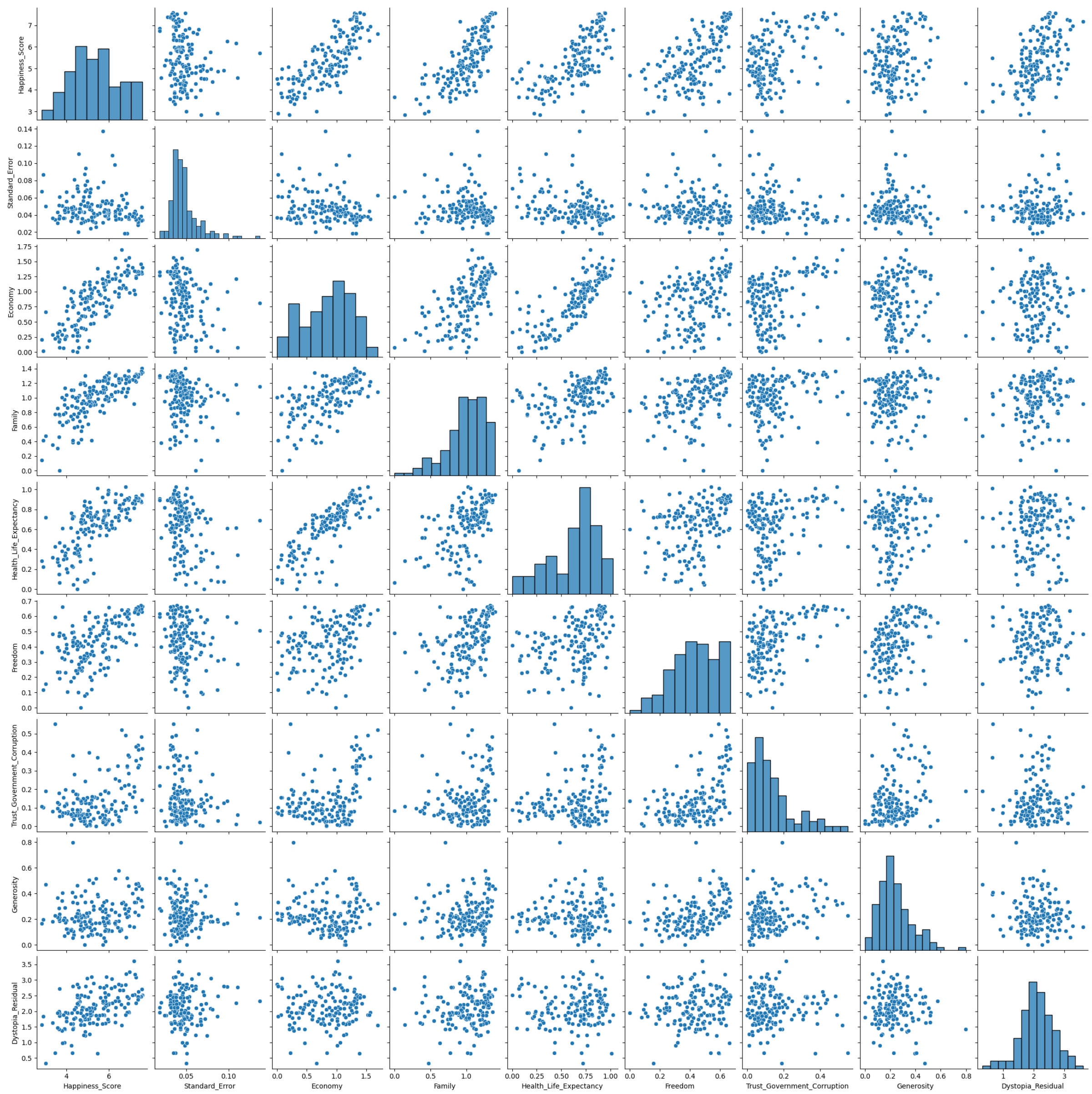
	Happiness_Score	Standard_Error	Economy	Family	Health_Life_Expectancy	Freedom	Trust_
0	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	
1	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	
2	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	
3	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	
4	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	

Observations:

I have removed the categorical columns 'Country,' 'Predicted_Happiness,' and 'Region,' as well as 'Happiness Rank,' which consisted of sequential numeric data.

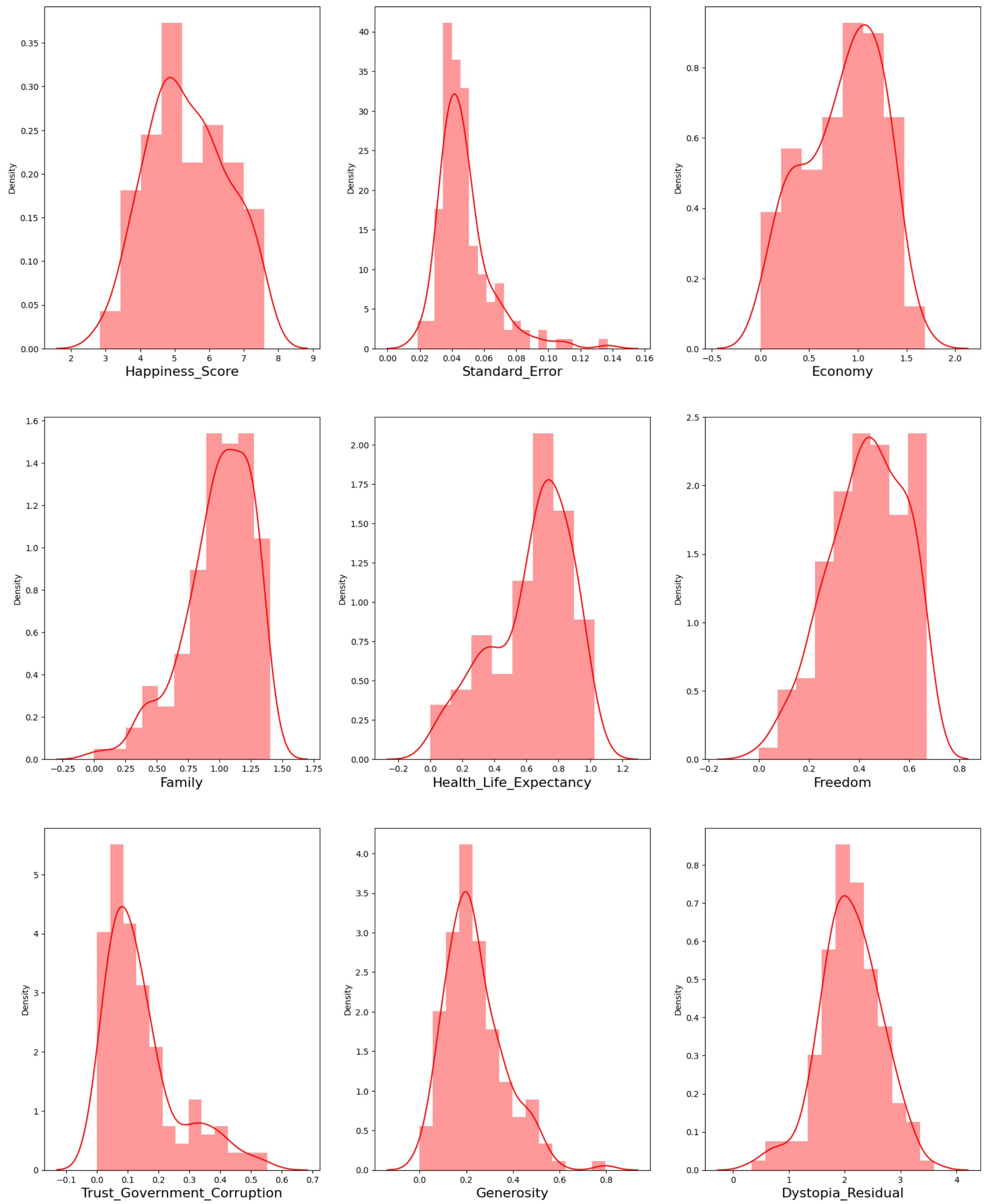
MultiVariate Analysis

In [19]:
```
sns.pairplot(data)
plt.show()
```



Skewness

In [20]:
```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber =1
for column in data.columns:
    if plotnumber <=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(df[column], color='r')
        plt.xlabel(column,fontsize=16)
    plotnumber+=1
plt.show()
```



Observations:

I can observe that there are columns displaying skewness and do not exhibit a normal distribution. However, I will validate this observation by utilizing the skewness function on the dataset.

In [21]:
```
data.skew()
```

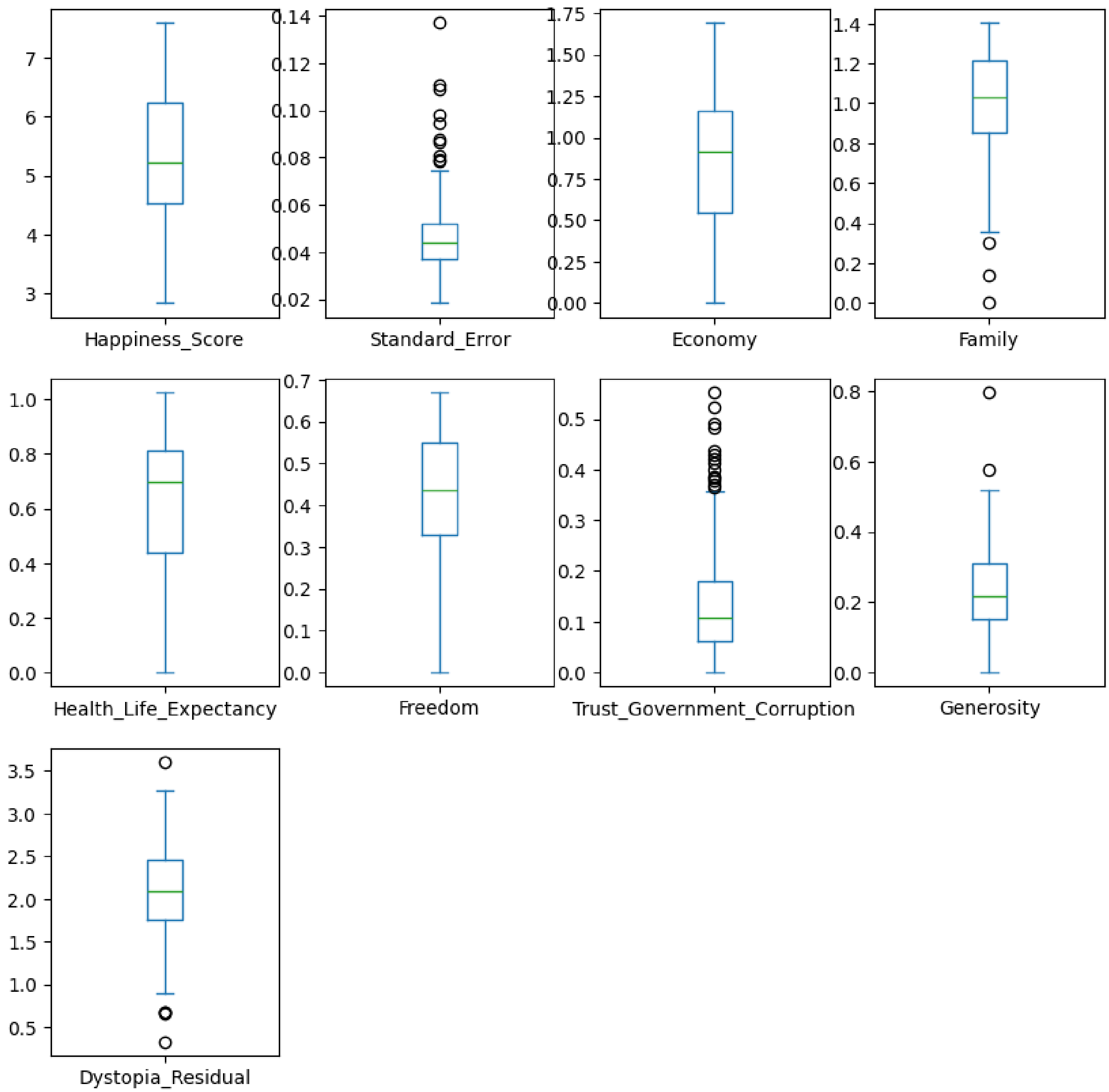
Out[21]:
```
Happiness_Score                 0.097769
Standard_Error                  1.983439
Economy                        -0.317575
Family                         -1.006893
Health_Life_Expectancy         -0.705328
Freedom                        -0.413462
Trust_Government_Corruption     1.385463
Generosity                      1.001961
Dystopia_Residual              -0.238911
dtype: float64
```

Observations:

- Happiness_Score: The skewness value of 0.097769suggests a slightly positive skewness, indicating a slightly longer right tail.
- Standard_Error: The skewness value of 1.983439 indicates a significant positive skewness, suggesting a long right tail and a concentration of lower values.
- Economy: The skewness value of -0.317575 suggests a slightly negative skewness, indicating a slightly longer left tail.
- Family: The skewness value of -1.006893 indicates a significant negative skewness, suggesting a long left tail and a concentration of higher values.
- Health_Life_Expectancy: The skewness value of -0.705328 indicates a negative skewness, suggesting a longer left tail.
- Freedom: The skewness value of -0.413462 suggests a slightly negative skewness, indicating a slightly longer left tail.
- Trust_Government_Corruption: The skewness value of 1.385463 indicates a significant positive skewness, suggesting a long right tail and a concentration of lower values.
- Generosity: The skewness value of 1.001961 suggests a positive skewness, indicating a longer right tail.
- Dystopia_Residual: The skewness value of -0.238911 suggests a slightly negative skewness, indicating a slightly longer left tail.

Outliers

In [22]:
```
data.plot(kind="box",subplots=True, layout=(3,4),figsize=(10,10))
plt.show()
```



Handling Outliers

In [23]:
```
#Importing library
from scipy.stats import zscore

# Z score method

z=np.abs(zscore(data))
threshold=3
np.where(z>3)

data1=data[(z<3).all(axis=1)]
data1
```

Out[23]:

	Happiness_Score	Standard_Error	Economy	Family	Health_Life_Expectancy	Freedom	Tru
0	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	
1	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	
2	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	
3	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	
4	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	
...	...	...	...	...	...	...	
150	3.655	0.05141	0.46534	0.77115	0.15185	0.46866	
151	3.587	0.04324	0.25812	0.85188	0.27125	0.39493	
152	3.575	0.03084	0.31982	0.30285	0.30335	0.23414	
154	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	
156	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	

149 rows × 9 columns

In [24]:
```
# Percentage of Data Loss

data_loss=(158-149)/158*100 # 158 was the number of rows in original data set a
data_loss
```

Out[24]: 5.69620253164557

Observations:

After removing the outliers we are checking the data loss percentage by comparing the rows in our original data set and the new data set after removing the outliers.

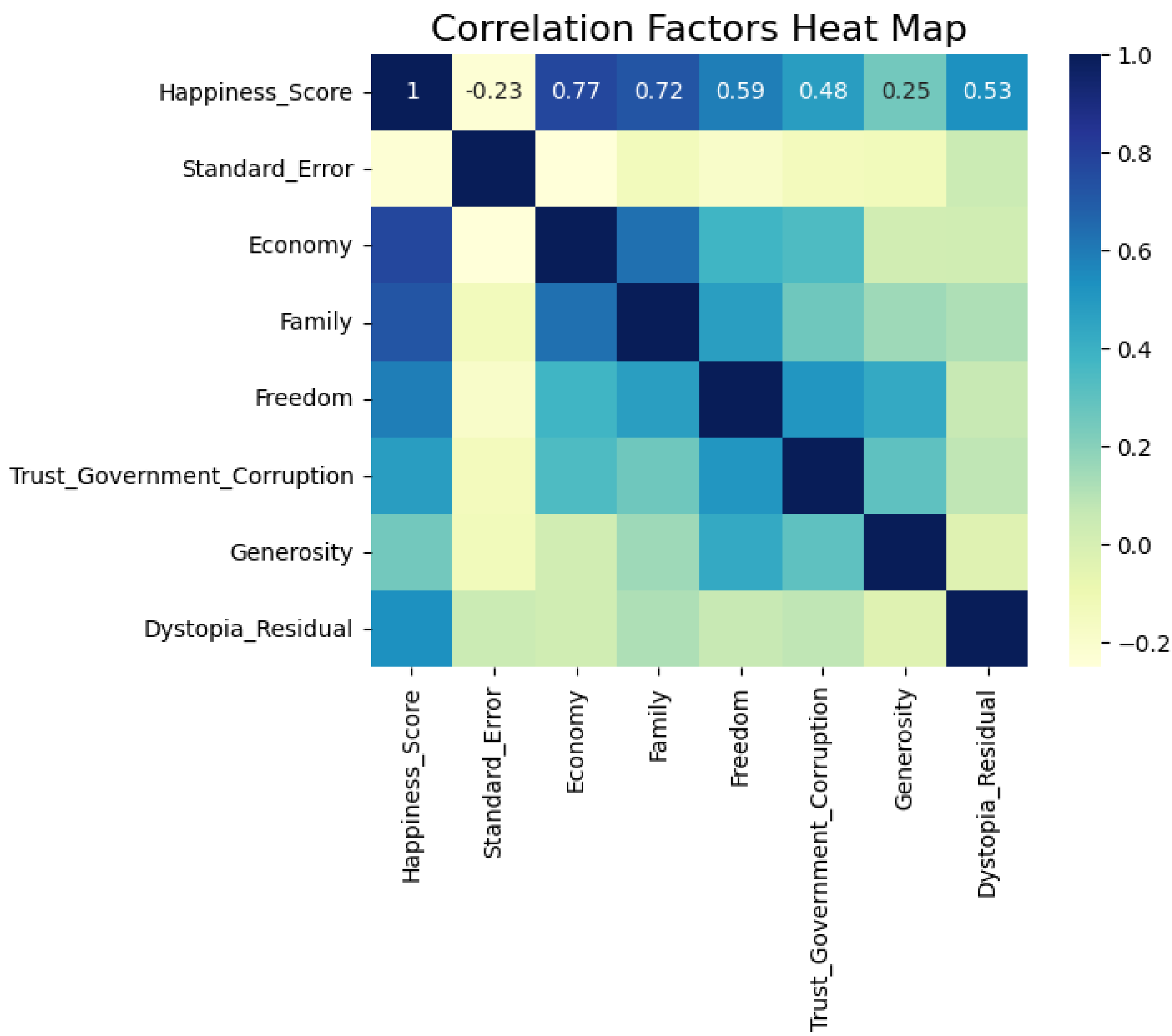
Checking correlation

In [25]:
```
data1.corr()
```

Out[25]:

	Happiness_Score	Standard_Error	Economy	Family	Health_Life_
Happiness_Score	1.000000	-0.230252	0.773577	0.720868	
Standard_Error	-0.230252	1.000000	-0.251749	-0.137879	
Economy	0.773577	-0.251749	1.000000	0.628589	
Family	0.720868	-0.137879	0.628589	1.000000	
Health_Life_Expectancy	0.729191	-0.356444	0.817470	0.503890	
Freedom	0.585066	-0.186465	0.376780	0.474229	
Trust_Government_Corruption	0.477692	-0.140156	0.342269	0.258646	
Generosity	0.250903	-0.131970	0.020730	0.154011	
Dystopia_Residual	0.528334	0.045722	0.026936	0.118062	

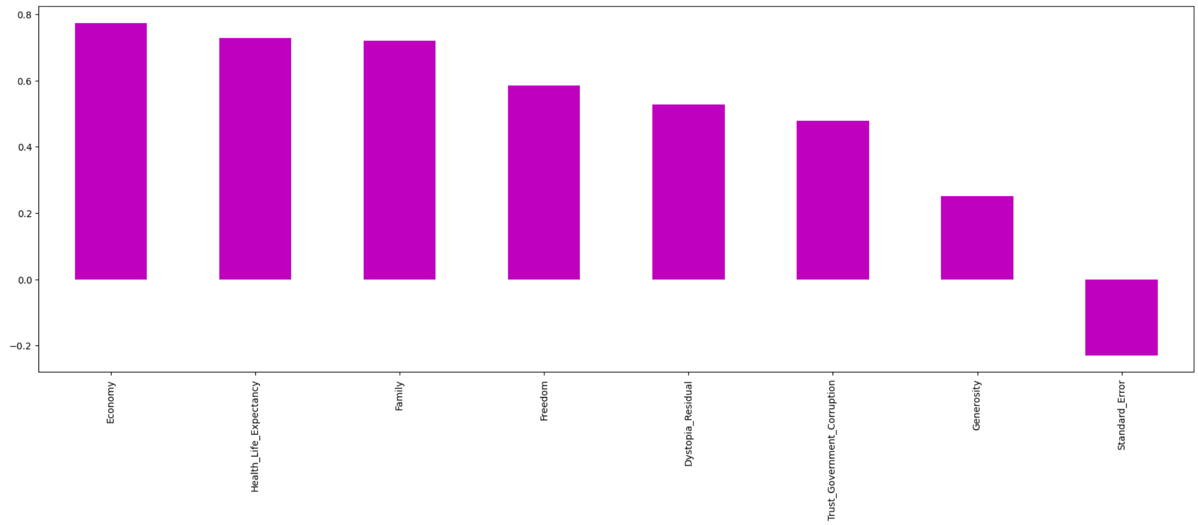
In [50]:
```
sns.heatmap(data1.corr(), annot =True, cmap ='YlGnBu').set_title('Correlation F
plt.show()
```



In [27]:
```
correlation_with_label = data1.corr()['Happiness_Score'].abs().sort_values(asce
print(correlation_with_label)
```

```
Happiness_Score                1.000000
Economy                        0.773577
Health_Life_Expectancy         0.729191
Family                         0.720868
Freedom                        0.585066
Dystopia_Residual              0.528334
Trust_Government_Corruption    0.477692
Generosity                     0.250903
Standard_Error                 0.230252
Name: Happiness_Score, dtype: float64
```

In [28]:
```
#Checking positive and Negative column

plt.figure(figsize=(22,7))
data1.corr()['Happiness_Score'].sort_values(ascending=False).drop(['Happiness_S
plt.xlabel=('Feature')
plt.ylabel=('column with target names')
plt.title=('correlation')
plt.show()
```



Observations:

All the columns are highly correlated with label

features correlation with features

In [29]:
```
df_corr = data1.corr()
correlation_threshold = 0.8
mask = df_corr.abs() >= correlation_threshold
features_to_drop = set()
for i in range(len(df_corr.columns)):
    for j in range(i+1, len(df_corr.columns)):
        if mask.iloc[i, j]:
            colname_i = df_corr.columns[i]
            colname_j = df_corr.columns[j]
            if colname_i not in features_to_drop:
                features_to_drop.add(colname_j)
features_to_drop
```

Out[29]:
```
{'Health_Life_Expectancy'}
```

Observation:

Here it is clear that the Health_Life_Expectancy is highly (more than 80 %) correlated with Economy so we will drop Health_Life_Expectancy

In [30]:
```
#Dropping the column
data1 = data1.drop(['Health_Life_Expectancy'], axis=1)
data1.head()
```

Out[30]:

	Happiness_Score	Standard_Error	Economy	Family	Freedom	Trust_Government_Corruption
0	7.587	0.03411	1.39651	1.34951	0.66557	0.41978
1	7.561	0.04884	1.30232	1.40223	0.62877	0.14145
2	7.527	0.03328	1.32548	1.36058	0.64938	0.48357
3	7.522	0.03880	1.45900	1.33095	0.66973	0.36503
4	7.427	0.03553	1.32629	1.32261	0.63297	0.32957

Spliting into feature and label

In [31]:
```
X = data1.drop('Happiness_Score', axis=1) # List of all features
y = data1['Happiness_Score'] # Data of our label
X
```

Out[31]:

	Standard_Error	Economy	Family	Freedom	Trust_Government_Corruption	Generosity	Dys
0	0.03411	1.39651	1.34951	0.66557	0.41978	0.29678	
1	0.04884	1.30232	1.40223	0.62877	0.14145	0.43630	
2	0.03328	1.32548	1.36058	0.64938	0.48357	0.34139	
3	0.03880	1.45900	1.33095	0.66973	0.36503	0.34699	
4	0.03553	1.32629	1.32261	0.63297	0.32957	0.45811	
...	...	...	...	...	...	...	
150	0.05141	0.46534	0.77115	0.46866	0.17922	0.20165	
151	0.04324	0.25812	0.85188	0.39493	0.12832	0.21747	
152	0.03084	0.31982	0.30285	0.23414	0.09719	0.36510	
154	0.03656	0.28665	0.35386	0.48450	0.08010	0.18260	
156	0.08658	0.01530	0.41587	0.11850	0.10062	0.19727	

149 rows × 7 columns

In [32]:
```
y
```

Out[32]:
```
0      7.587
1      7.561
2      7.527
3      7.522
4      7.427
       ...
150    3.655
151    3.587
152    3.575
154    3.340
156    2.905
Name: Happiness_Score, Length: 149, dtype: float64
```

Feature Scaling

In [33]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X= pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
X
```

Out[33]:

	Standard_Error	Economy	Family	Freedom	Trust_Government_Corruption	Generosity	D
0	-0.866786	1.381916	1.357879	1.583704	2.472255	0.546305	
1	0.185669	1.138324	1.567882	1.338953	0.009247	1.713898	
2	-0.926089	1.198220	1.401974	1.476027	3.036747	0.919630	
3	-0.531687	1.543526	1.283947	1.611371	1.987759	0.966495	
4	-0.765327	1.200315	1.250726	1.366887	1.673965	1.896418	
...	...	...	...	...	...	...	
144	0.369294	-1.026255	-0.945943	0.274090	0.343483	-0.249803	
145	-0.214450	-1.562163	-0.624365	-0.216276	-0.106943	-0.117411	
146	-1.100427	-1.402596	-2.811354	-1.285662	-0.382420	1.118051	
147	-0.691734	-1.488379	-2.608163	0.379439	-0.533653	-0.409226	
148	2.882182	-2.190139	-2.361154	-2.054764	-0.352067	-0.286458	

149 rows × 7 columns

Checking Best Random State

In [34]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_absolute_error
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
from xgboost import XGBRegressor

maxR2_score=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.33, ranc
    gb=GradientBoostingRegressor(n_estimators=100)
    gb.fit(X_train,y_train)
    y_pred=gb.predict(X_test)
    R2=r2_score(y_test,y_pred)
    if R2>maxR2_score:
        maxR2_score=R2
        maxRS=i
print('Best accuracy is', maxR2_score ,'on Random_state', maxRS)
```

```
Best accuracy is 0.9627310008619725 on Random_state 97
```

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  randc
```

Model Training & Testing

In [36]:
```
#Linear Regression
LR = LinearRegression()
LR.fit(X_train,y_train)
y_pred = LR.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9942668463219303
MAE 0.07201539628184
```

In [37]:
```
#Ridge Regression
R = Ridge(alpha=10)
R.fit(X_train,y_train)
y_pred = R.predict(X_test)

print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9915070860723721
MAE 0.08481780116591947
```

In [38]:
```
#Lasso Regression
L = Lasso(alpha=0.001)
L.fit(X_train,y_train)
y_pred = L.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9943446014982708
MAE 0.07128693838892541
```

In [39]:
```
#Decision Tree Regressor
DT = DecisionTreeRegressor(max_depth=5)
DT.fit(X_train,y_train)
y_pred = DT.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.8071770763537733
MAE 0.41377253968253985
```

In [40]:
```
#Random Forest Regressor
RF = RandomForestRegressor(n_estimators=100,
                             random_state=3,
                             max_samples=0.75,
                             max_features=0.75,
                             max_depth=10)
RF.fit(X_train,y_train)
y_pred = RF.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9234070092645572
MAE 0.241766110714286
```

In [41]:
```
#Extra Tree Regressor
ET = ExtraTreesRegressor(n_estimators=100,
                              random_state=3,
                              max_samples=0.5,
                              max_features=0.75,
                              max_depth=10,
                            bootstrap=True)

ET.fit(X_train,y_train)
y_pred = ET.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9276885736874863
MAE 0.24548270555555582
```

In [42]:
```
#Ada Boost Regressor
AB = AdaBoostRegressor(n_estimators=100,learning_rate=1.0)
AB.fit(X_train,y_train)
y_pred = AB.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.896808630330883
MAE 0.29237676465726675
```

In [43]:
```
#Gradient Boosting Regressor
GB = GradientBoostingRegressor(n_estimators=100)
GB.fit(X_train,y_train)
y_pred = GB.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9604686633422486
MAE 0.17305730682897288
```

In [44]:
```
#XGB Regressor
XG = XGBRegressor(n_estimators=50,max_depth=3,learning_rate=0.1)
XG.fit(X_train,y_train)
y_pred = XG.predict(X_test)
print('R2_score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
R2_score 0.9520709326025384
MAE 0.1911133590062459
```

DataFrame of all The models

In [45]:
```
models = {
    'Linear Regression': LR,
    'Ridge': R,
    'Lasso': L,
    'Decision Tree':DT,
    'Random Forest': RF,
    'Extra Trees': ET,
    'AdaBoost': AB,
    'Gradient Boosting': GB,
    'XGBoost': XG
}

results_df = pd.DataFrame(columns=['Model', 'R2 Score', 'MAE'])

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    results_df = results_df.append({'Model': model_name, 'R2 Score': r2, 'MAE':
    results_df_sorted = results_df.sort_values('R2 Score',ascending=False)

results_df_sorted
```

Out[45]:

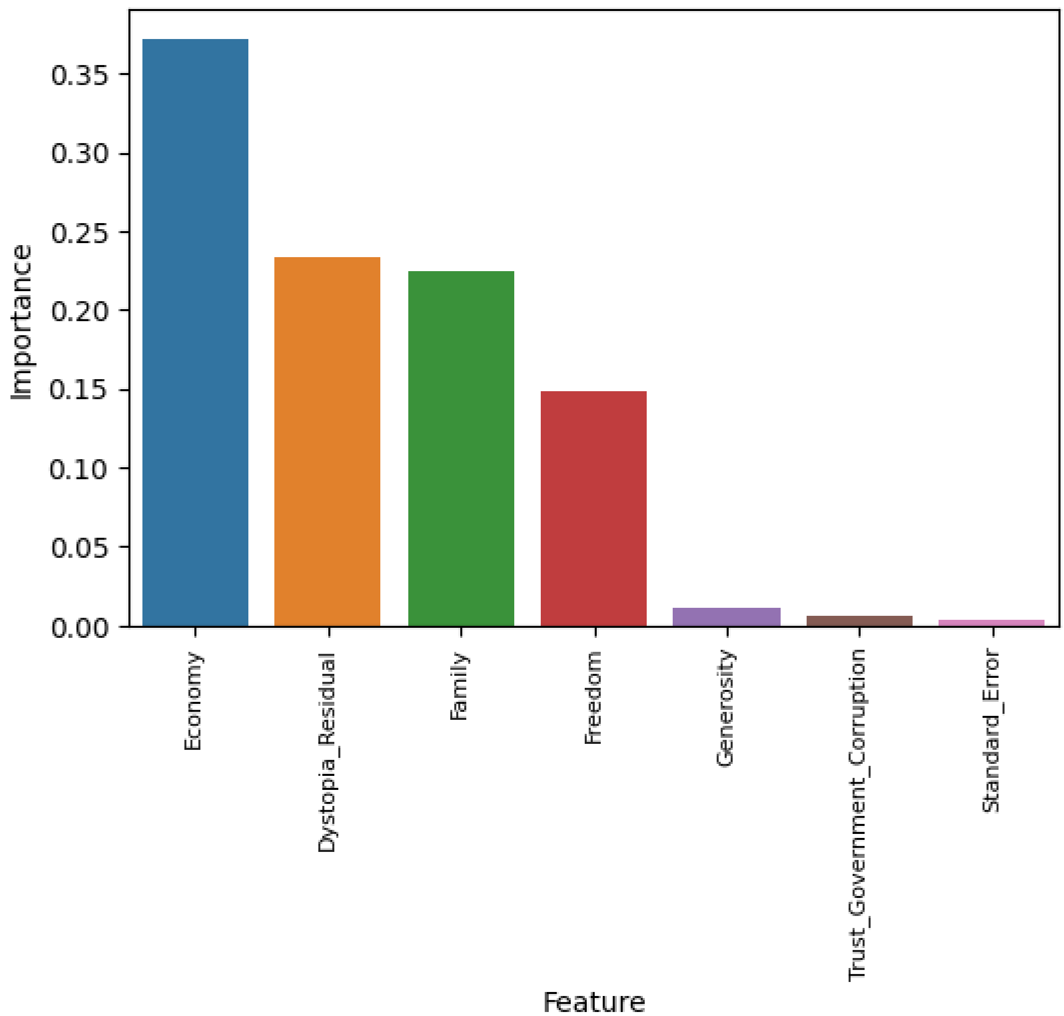
	Model	R2 Score	MAE
2	Lasso	0.994345	0.071287
0	Linear Regression	0.994267	0.072015
1	Ridge	0.991507	0.084818
7	Gradient Boosting	0.959341	0.178792
8	XGBoost	0.952071	0.191113
5	Extra Trees	0.927689	0.245483
4	Random Forest	0.923407	0.241766
6	AdaBoost	0.893432	0.306672
3	Decision Tree	0.810506	0.399056

Observation:

Here i can say that 'Linear Regression', 'Lasso', 'Ridge' are looking overfitted so the 'Gradient Boosting ' is the best model we will save it for further prediction.

Feature Importance

In [46]:
```
# Train the XGBoost model (I have already trained and named it 'XG')
GB.fit(X_train, y_train)

importances = GB.feature_importances_
feature_names = X_train.columns
feature_importances_df = pd.DataFrame({'Feature': feature_names, 'Importance':
feature_importances_df = feature_importances_df.sort_values('Importance', ascer

# Plotting the feature importances
plt.figure(figsize=(6,4))
sns.barplot(x='Feature', y='Importance', data=feature_importances_df)
plt.xticks(rotation=90, fontsize=8)
plt.show()
```



Observation:

Here we can see the importance of the feature in prediction

Saving the Model

In [47]:
```
import pickle

filename = 'world_happiness.pkl'
pickle.dump(GB, open(filename, 'wb'))
pickle.dump(scaler, open('scaler.pkl','wb'))
```

In [ ]:
```

```