Importing libraries and reading csv data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
from scipy.stats import zscore

df = pd.read_csv("data.csv", encoding="latin1")
df.columns = df.columns.str.lower().str.replace(" ","_")
df.head(10)
```

```
  country_name  year  life_ladder  log_gdp_per_capita  social_support
\
0  Afghanistan  2008        3.724               7.350           0.451

1  Afghanistan  2009        4.402               7.509           0.552

2  Afghanistan  2010        4.758               7.614           0.539

3  Afghanistan  2011        3.832               7.581           0.521

4  Afghanistan  2012        3.783               7.661           0.521

5  Afghanistan  2013        3.572               7.680           0.484

6  Afghanistan  2014        3.131               7.671           0.526

7  Afghanistan  2015        3.983               7.654           0.529

8  Afghanistan  2016        4.220               7.650           0.559

9  Afghanistan  2017        2.662               7.648           0.491


   healthy_life_expectancy_at_birth  freedom_to_make_life_choices
generosity  \
0                            50.500                         0.718
0.164
1                            50.800                         0.679
0.187
2                            51.100                         0.600
0.118
3                            51.400                         0.496
0.160
4                            51.700                         0.531
0.234
5                            52.000                         0.578
0.059
```

```
6                            52.300                         0.509
0.102
7                            52.600                         0.389
0.078
8                            52.925                         0.523
0.040
9                            53.250                         0.427
-0.123

   perceptions_of_corruption  positive_affect  negative_affect
0                      0.882            0.414            0.258
1                      0.850            0.481            0.237
2                      0.707            0.517            0.275
3                      0.731            0.480            0.267
4                      0.776            0.614            0.268
5                      0.823            0.547            0.273
6                      0.871            0.492            0.375
7                      0.881            0.491            0.339
8                      0.793            0.501            0.348
9                      0.954            0.435            0.371
```

```
print(df.info())
print(f"Dataset shape: {df.shape}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2363 entries, 0 to 2362
Data columns (total 11 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   country_name                      2363 non-null   object
 1   year                              2363 non-null   int64
 2   life_ladder                       2363 non-null   float64
 3   log_gdp_per_capita                2335 non-null   float64
 4   social_support                    2350 non-null   float64
 5   healthy_life_expectancy_at_birth  2300 non-null   float64
 6   freedom_to_make_life_choices      2327 non-null   float64
 7   generosity                        2282 non-null   float64
 8   perceptions_of_corruption         2238 non-null   float64
 9   positive_affect                   2339 non-null   float64
 10  negative_affect                   2347 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 203.2+ KB
None
Dataset shape: (2363, 11)
```

Checking the empty data and starting the cleaning.

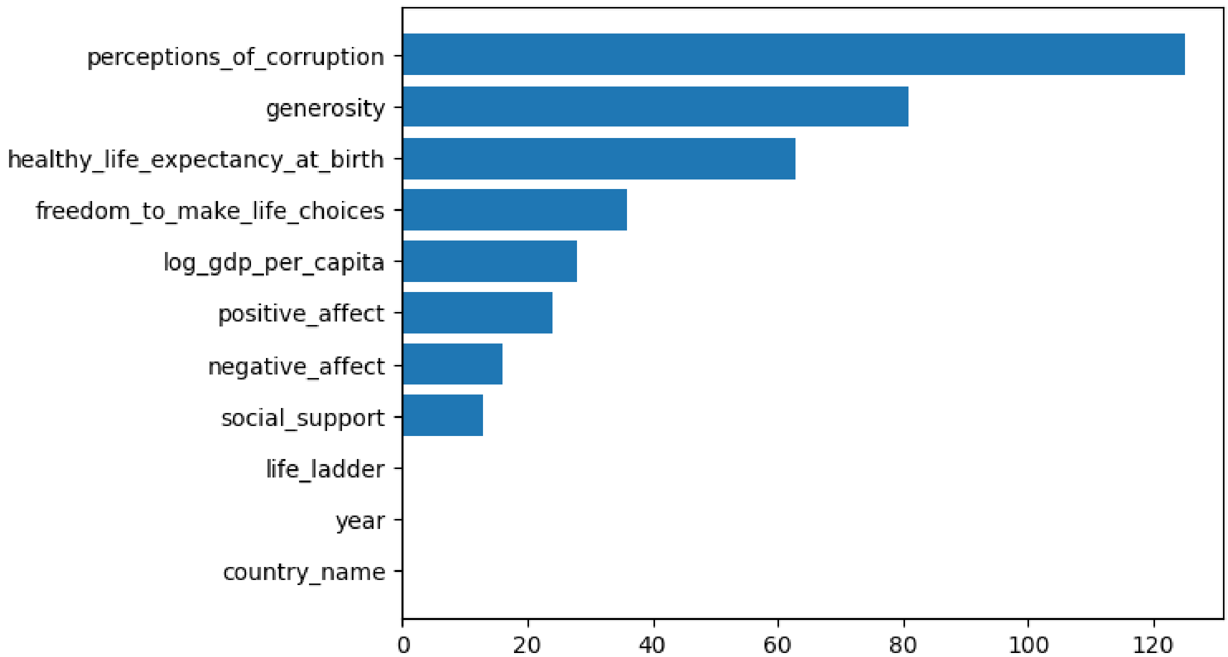
I deleted the rows with least empty values because of two reasons:

Reason #1: They werent likely to affect the results because how small their proportion is.

Reason #2: We cant do interpolation or data imputation because they werent continuous numerical measurement data but survey based responses.

```
p = df.isna().sum()
p = p.sort_values()
print(p)
print(f"\n\nPercentages: \n{df.isna().mean().sort_values() * 100}")

df = df.dropna(subset=["social_support", "negative_affect",
"positive_affect"])
row_count_df = df.country_name.value_counts(dropna=False)

plt.barh(p.index, p)
plt.show()
```

```
country_name                        0
year                                0
life_ladder                         0
social_support                     13
negative_affect                    16
positive_affect                    24
log_gdp_per_capita                 28
freedom_to_make_life_choices       36
healthy_life_expectancy_at_birth   63
generosity                         81
perceptions_of_corruption         125
dtype: int64


Percentages:
country_name                        0.000000
year                                0.000000
life_ladder                         0.000000
social_support                      0.550148
negative_affect                     0.677105
positive_affect                     1.015658
log_gdp_per_capita                  1.184934
freedom_to_make_life_choices        1.523487
healthy_life_expectancy_at_birth    2.666102
generosity                          3.427846
perceptions_of_corruption           5.289886
dtype: float64
```

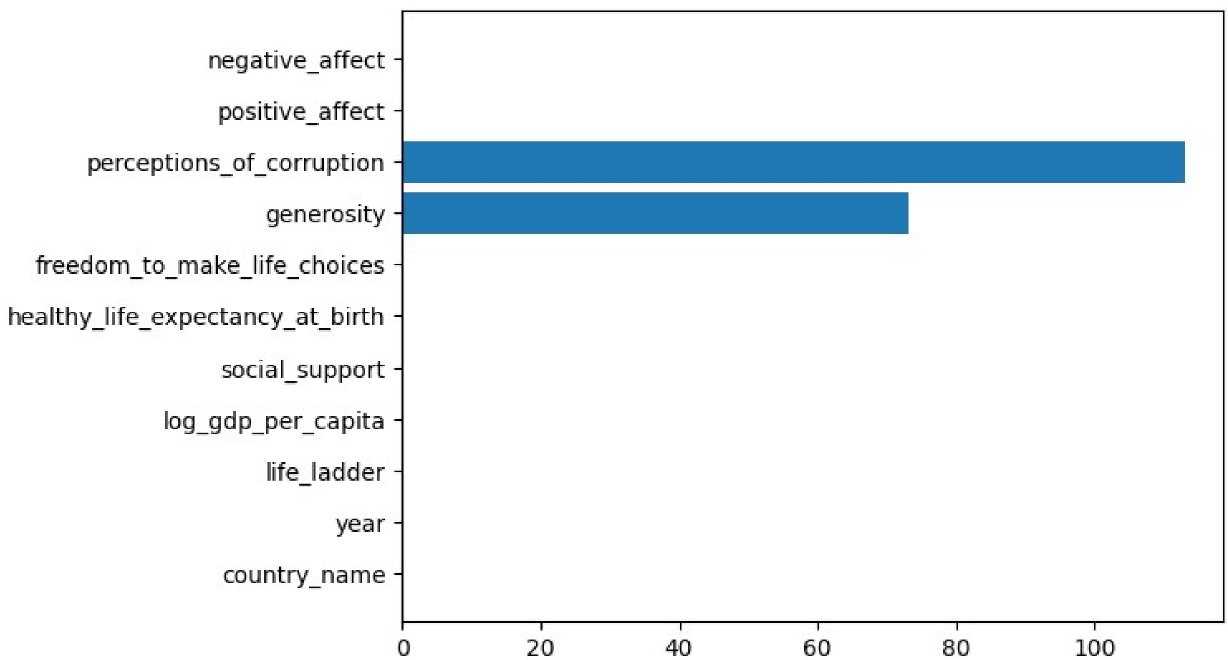


Since columns Log GDP per capita, Freedom to make life choices and Healthy life expectancy at birth are time dependent numerical measurements/computations so per country interpolation for each row with empty data is better here.

```
df = df.sort_values(["country_name", "year"])

cols_interp = [
    "log_gdp_per_capita",
    "freedom_to_make_life_choices",
    "healthy_life_expectancy_at_birth"
]

df[cols_interp] = (df.groupby("country_name")
[cols_interp].transform(lambda x: x.interpolate(method="linear",
limit_direction="both")))
df[cols_interp] = df[cols_interp].fillna(df[cols_interp].mean())

p = df.isna().sum()
print(p)
print(f"\n\nPercentages: \n{df.isna().mean().sort_values() * 100}")
plt.barh(p.index, p)
plt.show()
```

```
country_name                        0
year                                0
life_ladder                         0
log_gdp_per_capita                  0
social_support                      0
healthy_life_expectancy_at_birth    0
```

```
freedom_to_make_life_choices          0
generosity                           73
perceptions_of_corruption           113
positive_affect                       0
negative_affect                       0
dtype: int64


Percentages:
country_name                          0.000000
year                                  0.000000
life_ladder                           0.000000
log_gdp_per_capita                    0.000000
social_support                        0.000000
healthy_life_expectancy_at_birth      0.000000
freedom_to_make_life_choices          0.000000
positive_affect                       0.000000
negative_affect                       0.000000
generosity                            3.126338
perceptions_of_corruption             4.839400
dtype: float64
```



Assumimg countries with similar happiness/GDP scores likely have similar corruption levels. KNN assigns a new data point to the majority set within its neighbors.

```
imputer = KNNImputer(n_neighbors=5)
cols_interp = ["generosity", "perceptions_of_corruption"]
```

```
df_ = df[cols_interp]
df_ = imputer.fit_transform(df_)
df[cols_interp] = pd.DataFrame(df_, columns=cols_interp,
index=df.index)

p = df.isna().sum()
print(p)
print(f"\n\nPercentages: \n{df.isna().mean().sort_values() * 100}")
```

```
country_name                    0
year                            0
life_ladder                     0
log_gdp_per_capita              0
social_support                  0
healthy_life_expectancy_at_birth    0
freedom_to_make_life_choices    0
generosity                      0
perceptions_of_corruption       0
positive_affect                 0
negative_affect                 0
dtype: int64


Percentages:
country_name                    0.0
year                            0.0
life_ladder                     0.0
log_gdp_per_capita              0.0
social_support                  0.0
healthy_life_expectancy_at_birth    0.0
freedom_to_make_life_choices    0.0
generosity                      0.0
perceptions_of_corruption       0.0
positive_affect                 0.0
negative_affect                 0.0
dtype: float64
```

After we handled all missing data situation, here is a chart to see how many rows of data for each country (im not sure about what that error suppose to mean):

```
n_cols = 4
chunks = np.array_split(row_count_df, n_cols)

fig, axes = plt.subplots(1, n_cols, figsize=(16, 10))

for i, chunk in enumerate(chunks):
    ax = axes[i]
    ax.axis("off")

    table = pd.plotting.table(ax, chunk, loc="center",
```

```
cellLoc="center")

plt.tight_layout()
plt.show()
```

```
d:\projects\asil\.venv\Lib\site-packages\numpy\_core\
fromnumeric.py:57: FutureWarning: 'Series.swapaxes' is deprecated and
will be removed in a future version. Please use 'Series.transpose'
instead.
  return bound(*args, **kwds)
```

	count
Cameroon	18
Bolivia	18
Argentina	18
Chile	18
Colombia	18
Brazil	18
Costa Rica	18
Russia	18
Senegal	18
Uganda	18
Uruguay	18
Ukraine	18
Thailand	18
Tanzania	18
Mexico	18
Moldova	18
Lebanon	18
Kenya	18
Lithuania	18
Italy	18
Israel	18
Japan	18
Kazakhstan	18
Kyrgyzstan	18
India	18
Ghana	18
Indonesia	18
Georgia	18
Germany	18
France	18
El Salvador	18
Egypt	18
Denmark	18
Ecuador	18
Dominican Republic	18
South Africa	18
South Korea	18
Spain	18
Sweden	18
Tajikistan	18
United Kingdom	18

	count
Zimbabwe	18
Venezuela	18
Saudi Arabia	18
Peru	18
Nepal	18
Nicaragua	18
New Zealand	17
Panama	17
Türkiye	17
Sri Lanka	17
Uzbekistan	17
Zambia	17
Netherlands	17
Paraguay	17
Romania	17
Latvia	17
Armenia	17
Bangladesh	17
Australia	17
China	17
Canada	17
Belgium	17
United States	17
Poland	17
Pakistan	17
Philippines	17
Greece	17
Hungary	17
Kosovo	17
Honduras	17
Estonia	17
Mali	17
Ireland	17
Serbia	16
North Macedonia	16
Nigeria	16
Vietnam	16
Taiwan Province of China	16
State of Palestine	16
Malaysia	16
Iran	16

	count
Burkina Faso	16
Austria	16
Azerbaijan	16
Bosnia and Herzegovina	16
Chad	16
Cambodia	16
Albania	16
Slovenia	16
Singapore	16
Portugal	16
Niger	16
Cyprus	16
Finland	16
Mongolia	16
Guatemala	16
Croatia	16
Afghanistan	15
Malta	15
Malawi	15
Mauritania	15
Montenegro	15
Czechia	15
Benin	15
Sierra Leone	15
Slovakia	15
Bulgaria	15
Yemen	14
Botswana	14
Congo (Brazzaville)	14
Tunisia	14
United Arab Emirates	14
Iraq	14
Jordan	14
Switzerland	13
Belarus	13
Luxembourg	13
Hong Kong S.A.R. of China	13
Gabon	13
Madagascar	13
Guinea	13
Norway	13

	count
Myanmar	12
Togo	12
Morocco	12
Rwanda	12
Laos	12
Kuwait	12
Ivory Coast	12
Bahrain	11
Liberia	11
Haiti	11
Ethiopia	11
Iceland	11
Mozambique	11
Mauritius	10
Congo (Kinshasa)	10
Algeria	10
Turkmenistan	10
Jamaica	9
Namibia	9
Libya	8
Comoros	8
Syria	7
Central African Republic	5
Burundi	5
Lesotho	5
Gambia	5
Trinidad and Tobago	5
Sudan	5
South Sudan	4
Somaliland region	4
Angola	4
Eswatini	4
Somalia	3
Bhutan	3
Djibouti	3
Qatar	3
Belize	2
Guyana	1
Cuba	1
Suriname	1

Checking the data

```
df.describe().T
```

```
                                  count         mean       std
min  \
year                             2335.0  2014.806424  5.051089
2005.000
life_ladder                      2335.0     5.484140  1.126639
1.281
log_gdp_per_capita               2335.0     9.391299  1.162615
5.527
social_support                   2335.0     0.809576  0.121313
0.228
healthy_life_expectancy_at_birth 2335.0    63.428057  6.790525
```

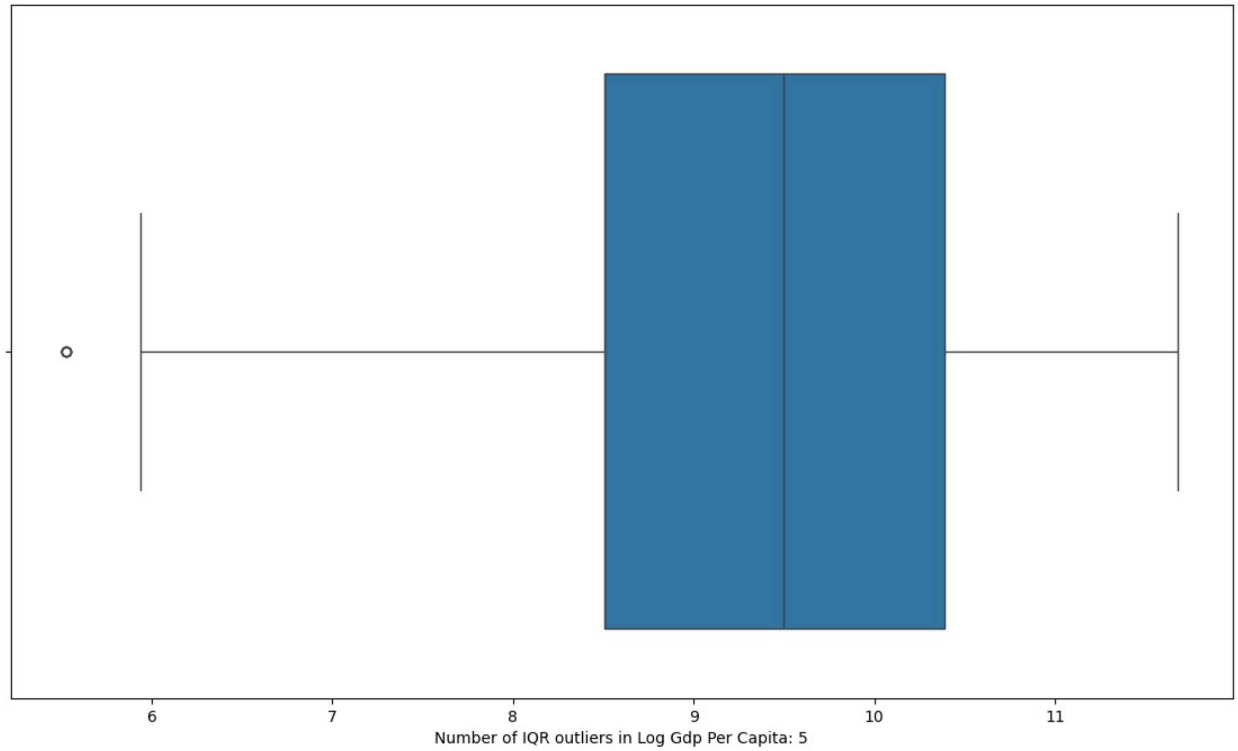
```
6.720
freedom_to_make_life_choices       2335.0     0.750375  0.139750
0.228
generosity                         2335.0     0.000709  0.159444    -
0.340
perceptions_of_corruption          2335.0     0.743834  0.182003
0.035
positive_affect                    2335.0     0.651798  0.106175
0.179
negative_affect                    2335.0     0.273077  0.087114
0.083

                                        25%       50%        75%
max
year                               2011.000  2015.000  2019.0000
2023.000
life_ladder                           4.647     5.451     6.3235
8.019
log_gdp_per_capita                    8.504     9.498    10.3900
11.676
social_support                        0.744     0.835     0.9040
0.987
healthy_life_expectancy_at_birth     59.490    65.000    68.5600
74.600
freedom_to_make_life_choices          0.661     0.772     0.8620
0.985
generosity                           -0.108    -0.020     0.0920
0.700
perceptions_of_corruption             0.687     0.797     0.8650
0.983
positive_affect                       0.572     0.663     0.7370
0.884
negative_affect                       0.209     0.262     0.3260
0.705
```

Checking Outliers:

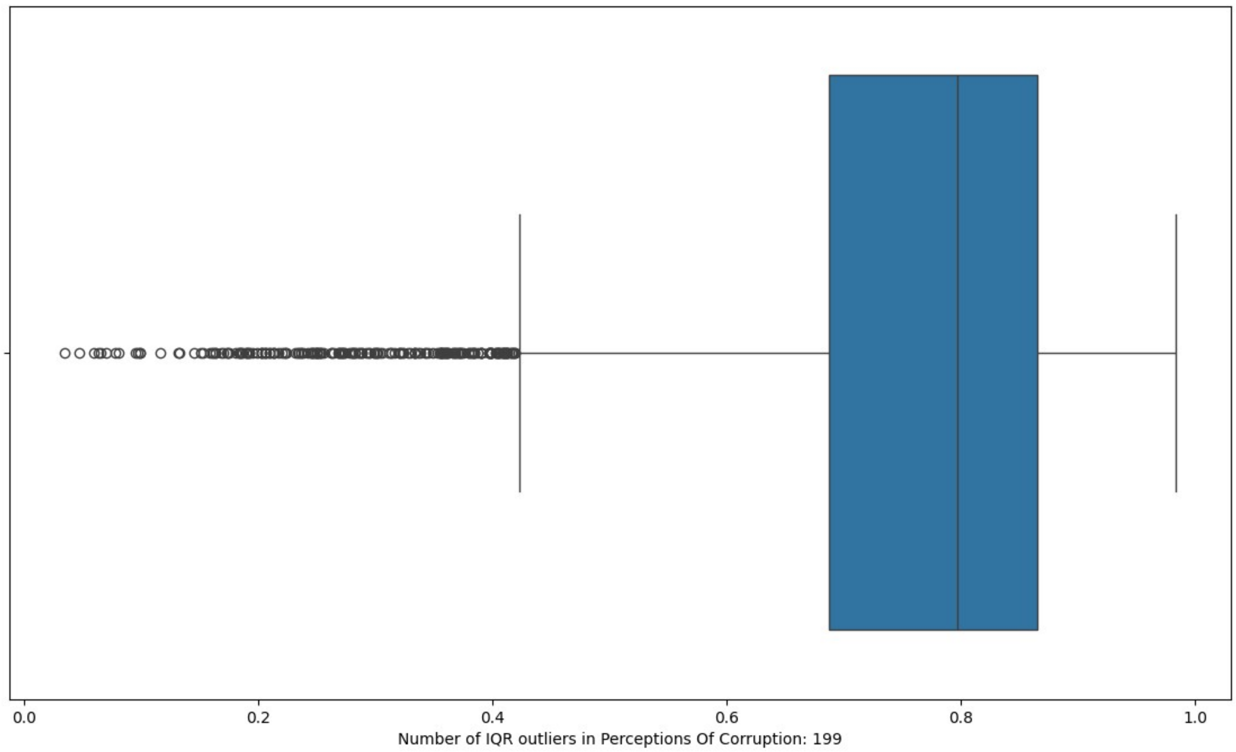
Because this data does not have to be normally distributed, i chose not to remove outliers.

```
criteria = df.log_gdp_per_capita

q1 = criteria.quantile(0.25)
q3 = criteria.quantile(0.75)
iqr = q3 - q1

outliers = df[(criteria < q1 - 1.5*iqr) | (criteria > q3 + 1.5*iqr)]
plt.figure(figsize=(14,8))
sns.boxplot(x=criteria)
plt.xlabel(f"Number of IQR outliers in
```

```
{str(criteria.name).replace("_", " ").title()}: {outliers.shape[0]}")
plt.show()
```

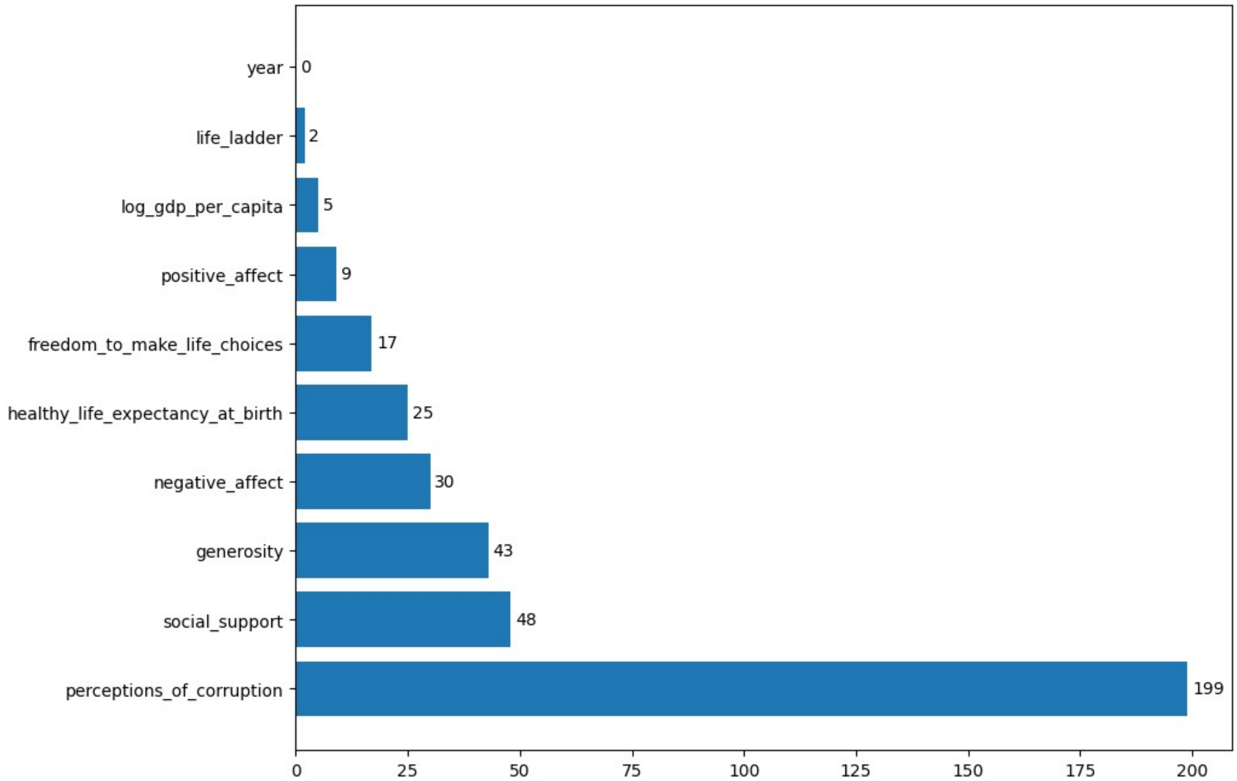



```
criteria = df.perceptions_of_corruption

q1 = criteria.quantile(0.25)
q3 = criteria.quantile(0.75)
iqr = q3 - q1

outliers = df[(criteria < q1 - 1.5*iqr) | (criteria > q3 + 1.5*iqr)]
plt.figure(figsize=(14,8))
sns.boxplot(x=criteria)
plt.xlabel(f"Number of IQR outliers in
{str(criteria.name).replace("_", " ").title()}: {outliers.shape[0]}")
plt.show()
```




```
num_cols = df.select_dtypes(include=np.number)

outlier_summary = {}

for col in num_cols.columns:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr

    outlier_count = df[(df[col] < lower) | (df[col] > upper)].shape[0]
    outlier_summary[col] = outlier_count

outlier_df = pd.DataFrame.from_dict(outlier_summary, orient="index",
columns=["outlier_count"]).sort_values("outlier_count",
ascending=False)
plt.figure(figsize=(10,8))
plt.barh(outlier_df.index, outlier_df.outlier_count)

for i, v in enumerate(outlier_df.outlier_count):
    plt.text(v + 1, i, str(v), va="center")


plt.show()
```

```python
print("IQR outliers:::")
for x in outlier_summary:
    print(f"{x.capitalize().replace("_", " ")}: {outlier_summary[x]}")
```

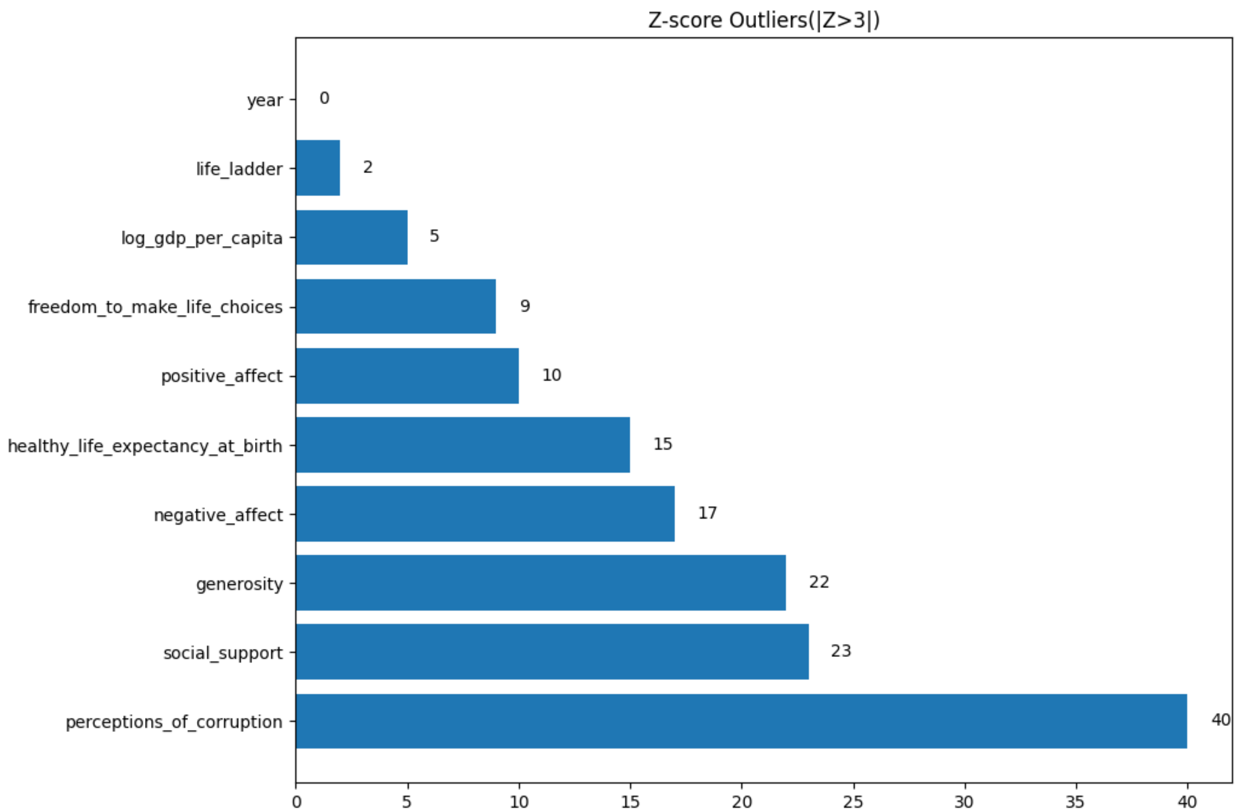



```
IQR outliers:::
Year: 0
Life ladder: 2
Log gdp per capita: 5
Social support: 48
Healthy life expectancy at birth: 25
Freedom to make life choices: 17
Generosity: 43
Perceptions of corruption: 199
Positive affect: 9
Negative affect: 30
```

Here is another way to determine the outliers: The Z-score, though it usually useful when data is continuous.

```python
outlier_summary = {}

for col in num_cols.columns:
    z = zscore(df[col], nan_policy="omit")
    outlier_count = np.sum(np.abs(z) > 3)
```

```python
    outlier_summary[col] = outlier_count

outlier_df = pd.DataFrame.from_dict(outlier_summary, orient="index",
columns=["outlier_count"]).sort_values("outlier_count",
ascending=False)
plt.figure(figsize=(10,8))
plt.barh(outlier_df.index, outlier_df.outlier_count)

for i, v in enumerate(outlier_df.outlier_count):
    plt.text(v + 1, i, str(v), va="center")

plt.title("Z-score Outliers(|Z>3|)")
plt.show()

print(outlier_df)
```

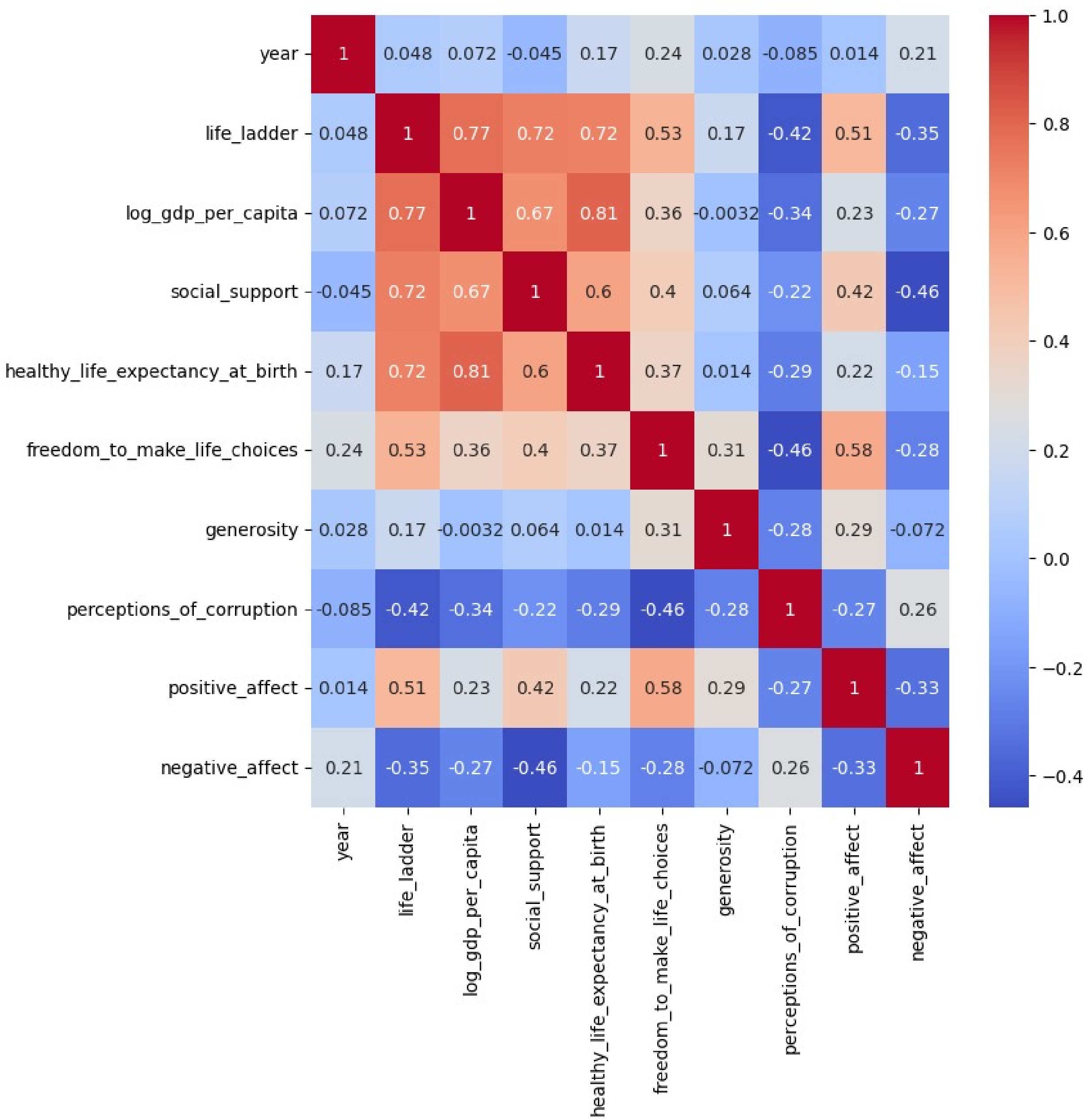



```
                                  outlier_count
perceptions_of_corruption                    40
social_support                               23
generosity                                   22
negative_affect                              17
healthy_life_expectancy_at_birth             15
positive_affect                              10
freedom_to_make_life_choices                  9
```

```
log_gdp_per_capita                              5
life_ladder                                     2
year                                            0
```

Correlation matrix

```
num_cols = df.select_dtypes(include=[np.number])
corr_matrix = num_cols.corr()

plt.figure(figsize=(8,8))
sns.heatmap(data=corr_matrix, cmap="coolwarm", annot=True)

mask = np.triu(np.ones(corr_matrix.shape)).astype(bool)
l1 = corr_matrix.mask(mask)
plt.show()
```



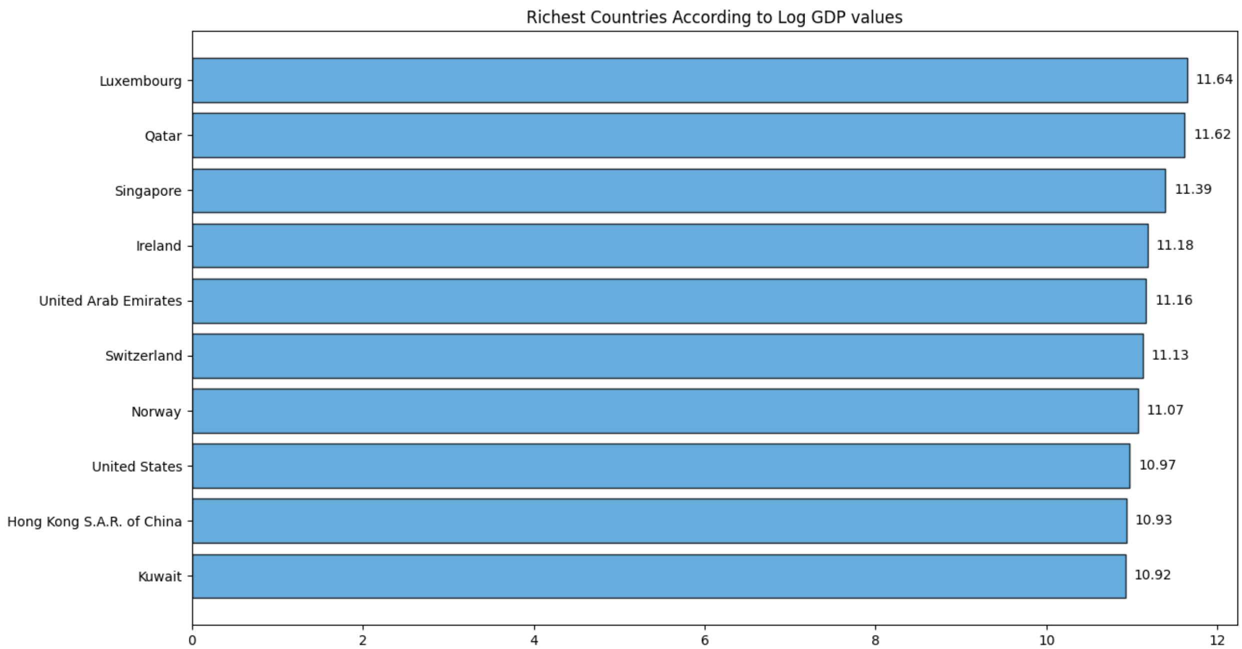
As shown in the correlation matrix:

- Happiness and Wealth: As a country gets richer (Log GDP per capita), people's happiness levels (Life Ladder) increase very strongly (0.77 correlation).
- Health and Wealth: The strongest relationship in the dataset is between economic prosperity and healthy life expectancy; citizens of wealthy countries live significantly longer and healthier lives (0.81 correlation).
- Social Support and Happiness: Having someone to rely on during difficult times (Social Support) greatly increases overall life satisfaction (Life Ladder) (0.72 correlation).

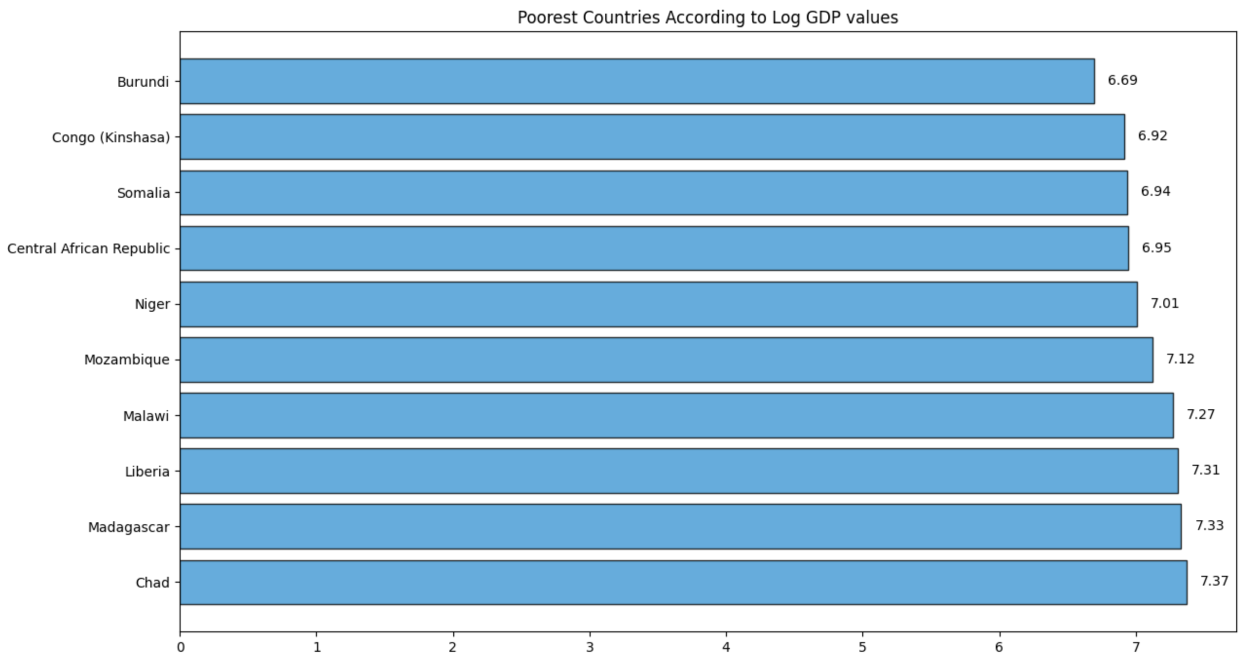
- Freedom and Positive Emotions: Having the freedom to make choices about one's life directly increases the positive emotions (joy, laughter, etc.) felt during the day (0.58 correlation).
- Corruption and Happiness: As the perception of corruption in a country increases, overall happiness levels and life satisfaction weakly decreases (-0.42 correlation).
- Generosity and Happiness: Surprisingly, there is a very weak relationship between generosity and overall happiness (Life Ladder), meaning that a generous society is not necessarily a happy one (0.17 correlation).
- Freedom and Wealth: As a country gets richer (Log GDP per capita), people's sense of freedom to make choices about their own lives also increases. But it's weakly correlated (0.36 correlation).
- Corruption and Freedom: In countries with high perceptions of corruption, people's freedom to make choices about their own lives decreases (-0.46 correlation).

Most and Least Rich Countries According to Log GDP

```
modded_df = df.groupby("country_name", as_index=False).median()

plot1 = modded_df[["country_name","log_gdp_per_capita"]]
plot1 = plot1.sort_values("log_gdp_per_capita", ascending=False)
topn = 10
topx = plot1.country_name.head(topn)
topy = plot1.log_gdp_per_capita.head(topn)

plt.figure(figsize=(14, 8))
plt.barh(topx, topy, color="#4097d4", edgecolor="black", alpha=0.8)
plt.gca().invert_yaxis()
for y, x in enumerate(topy):
    plt.annotate(f"{x:.2f}", xy=(x+0.1, y), va="center")
plt.title("Richest Countries According to Log GDP values")
plt.show()
```



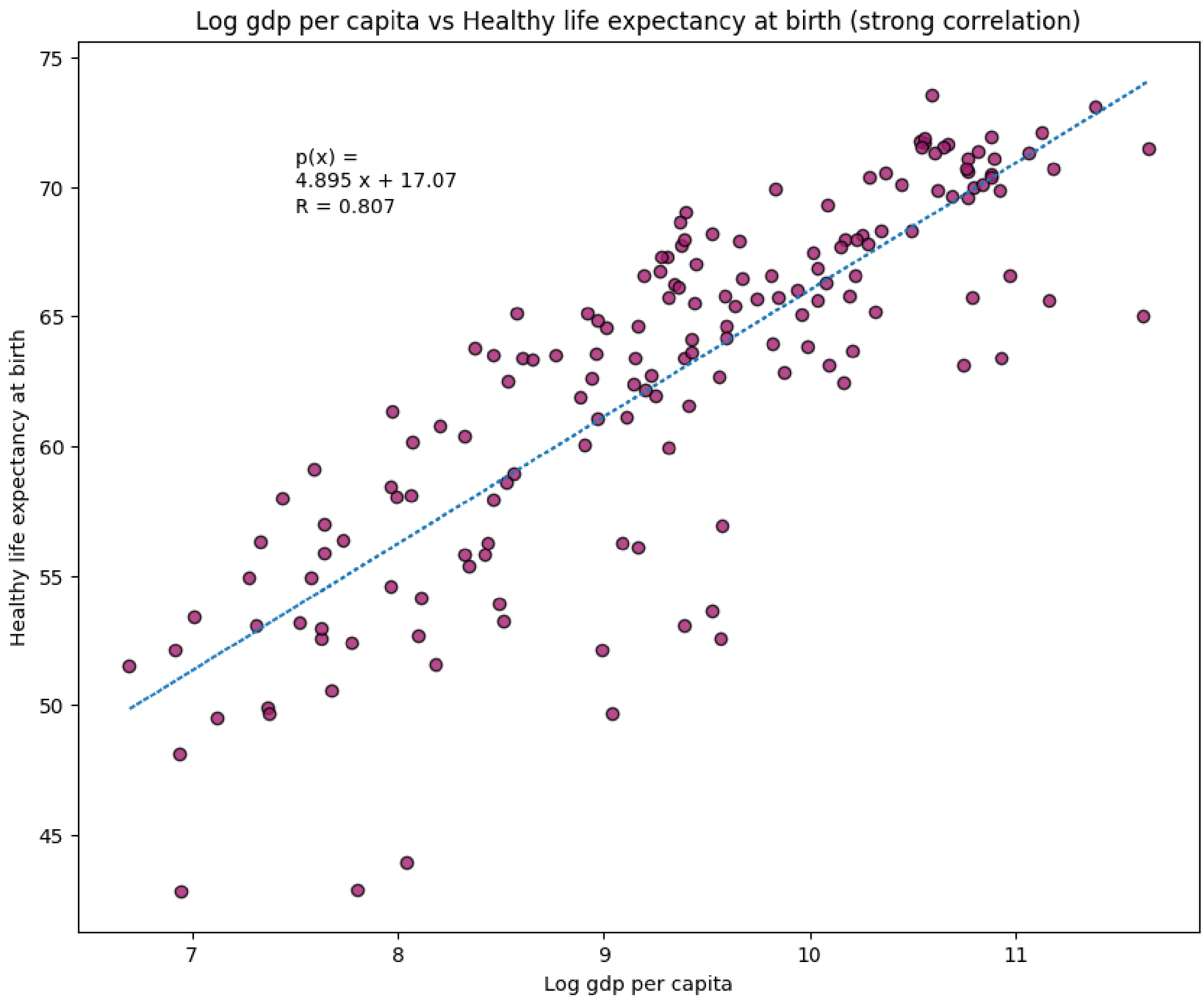

```
plot1 = modded_df[["country_name","log_gdp_per_capita"]]
plot1=plot1.sort_values("log_gdp_per_capita", ascending=True)
topn = 10
topx = plot1.country_name.head(topn)
topy = plot1.log_gdp_per_capita.head(topn)

plt.figure(figsize=(14, 8))
plt.barh(topx, topy, color="#4097d4", edgecolor="black", alpha=0.8)
plt.gca().invert_yaxis()
for y, x in enumerate(topy):
    plt.annotate(f"{x:.2f}", xy=(x+0.1, y), va="center")
plt.title("Poorest Countries According to Log GDP values")
plt.show()
```



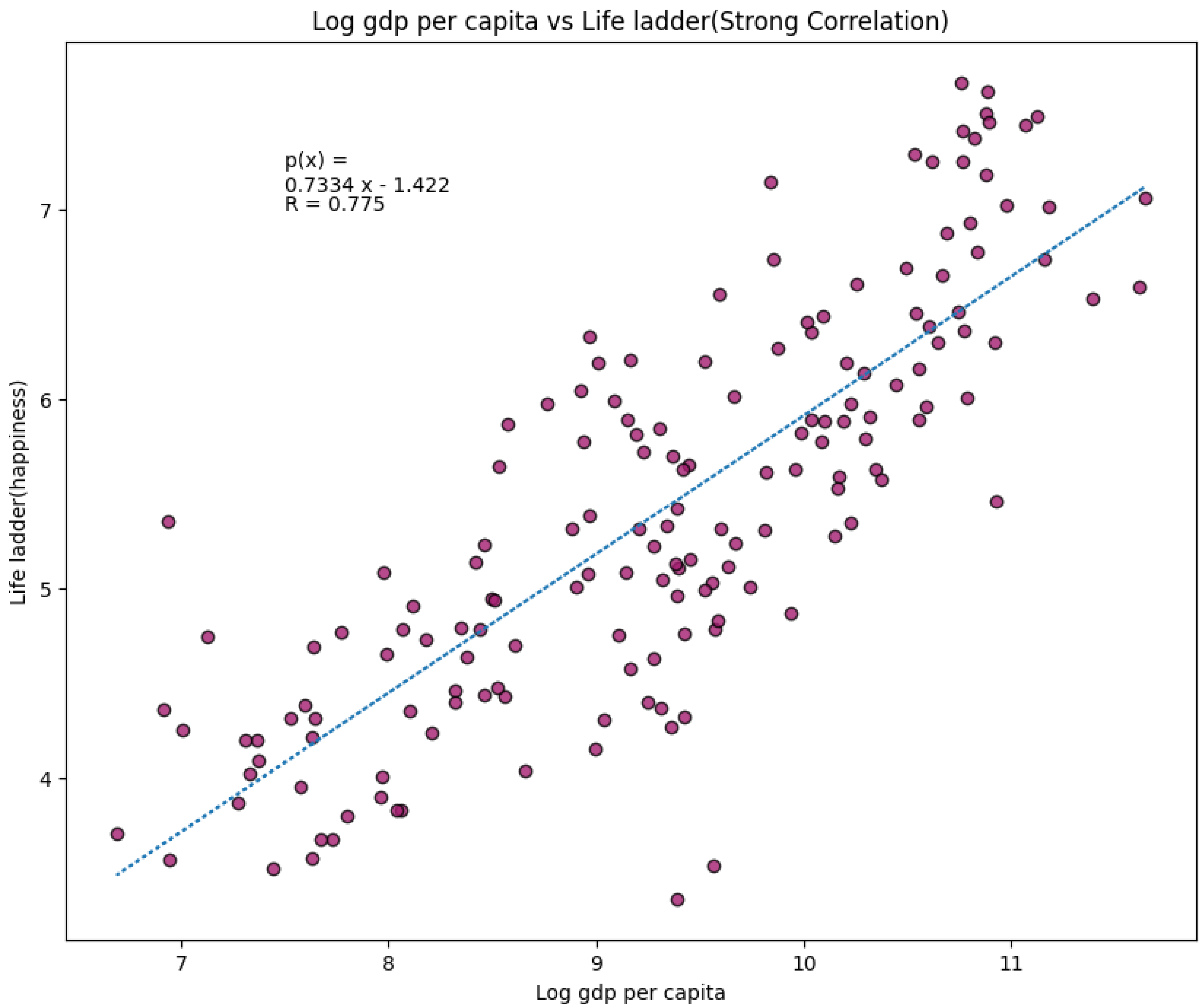
There is a strong correlation between healthy life expectancy at birth and log GDP per capita (r = 0.81), so we visualized their relationship using a scatter plot. As shown in the plot, healthy life expectancy tends to increase as log GDP per increases.

We can add a trendline to better show the correlation

```
valid = modded_df[["log_gdp_per_capita",
"healthy_life_expectancy_at_birth"]]

plt.figure(figsize=(10, 8))
plt.scatter(valid.log_gdp_per_capita,
valid.healthy_life_expectancy_at_birth, color="#A41D70",
edgecolor="black", alpha=0.8)
plt.ylabel("Healthy life expectancy at birth")
plt.xlabel("Log gdp per capita")
plt.title("Log gdp per capita vs Healthy life expectancy at birth
(strong correlation)")

z = np.polyfit(valid.log_gdp_per_capita,
valid.healthy_life_expectancy_at_birth, 1)
p = np.poly1d(z)

plt.plot(valid.log_gdp_per_capita,
p(valid.log_gdp_per_capita),linestyle = "dotted")
plt.annotate(f"p(x) = {p}", (7.5,70))
plt.annotate(f"R =
{l1.log_gdp_per_capita.healthy_life_expectancy_at_birth:.3f}",
(7.5,69))

plt.show()
```

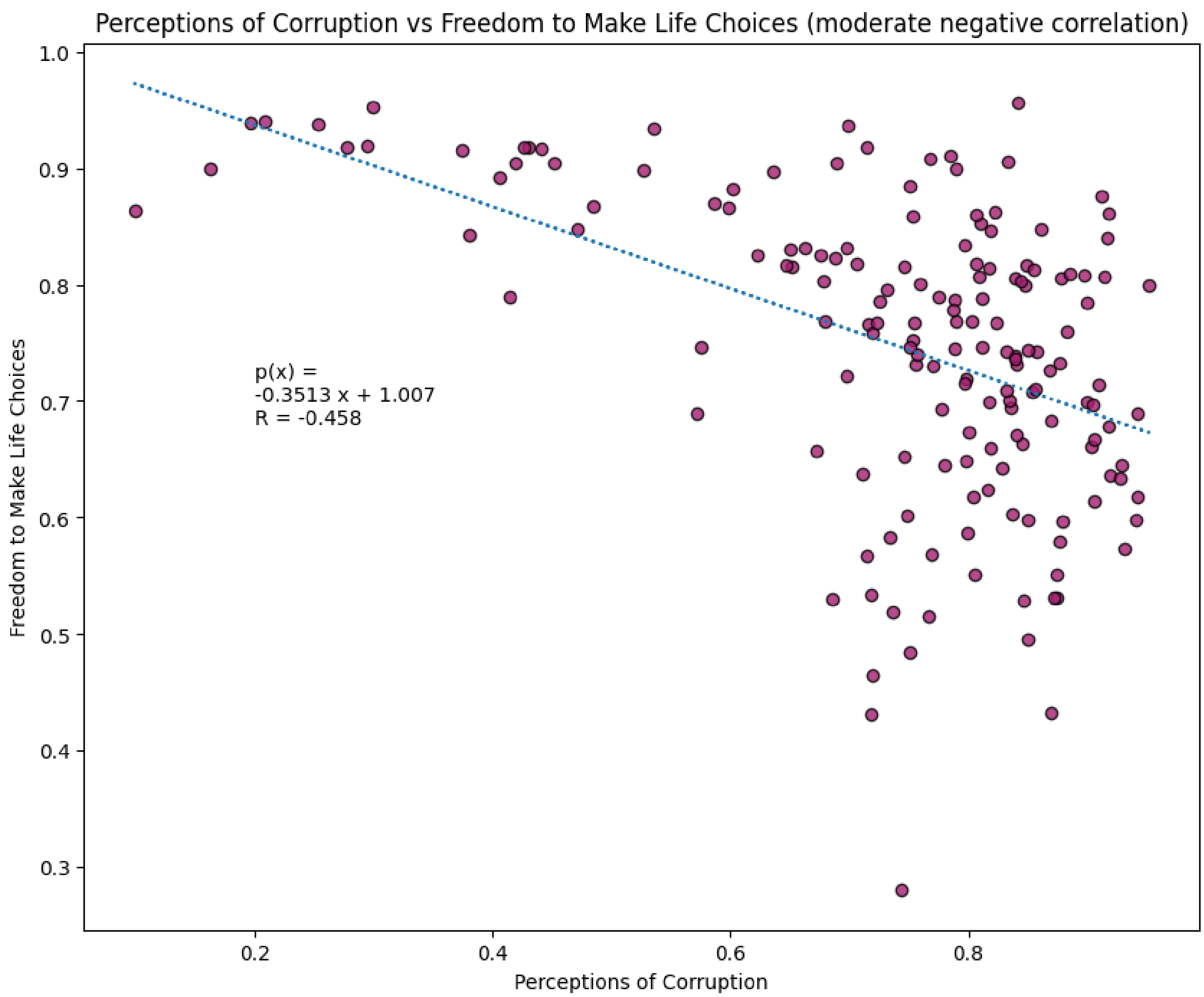



```
valid = modded_df[["log_gdp_per_capita", "life_ladder"]]

plt.figure(figsize=(10, 8))
plt.scatter(valid.log_gdp_per_capita, valid.life_ladder,
color="#A41D70", edgecolor="black", alpha=0.8)
plt.ylabel("Life ladder(happiness)")
plt.xlabel("Log gdp per capita")
plt.title("Log gdp per capita vs Life ladder(Strong Correlation)")

z = np.polyfit(valid.log_gdp_per_capita, valid.life_ladder, 1)
p = np.poly1d(z)

plt.plot(valid.log_gdp_per_capita,
p(valid.log_gdp_per_capita),linestyle = "dotted")
plt.annotate(f"p(x) = {p}", (7.5,7.1))
plt.annotate(f"R = {l1.life_ladder.log_gdp_per_capita:.3f}", (7.5,7))

plt.show()
```



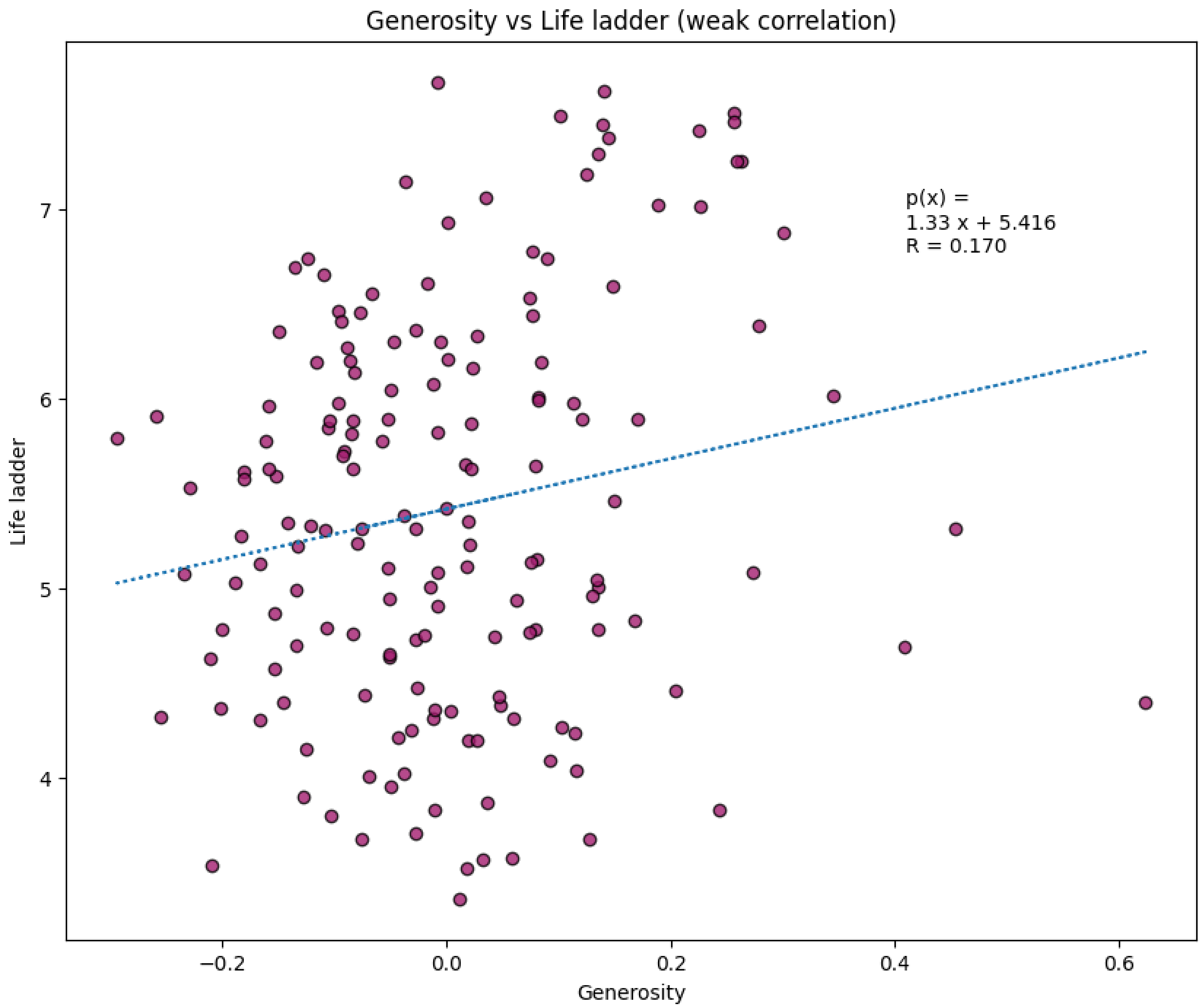

```
valid = modded_df[["perceptions_of_corruption",
"freedom_to_make_life_choices"]]

plt.figure(figsize=(10, 8))
plt.scatter(valid.perceptions_of_corruption,
valid.freedom_to_make_life_choices, color="#A41D70",
edgecolor="black", alpha=0.8)
plt.ylabel("Freedom to Make Life Choices")
plt.xlabel("Perceptions of Corruption")
plt.title("Perceptions of Corruption vs Freedom to Make Life Choices
(moderate negative correlation)")

z = np.polyfit(valid.perceptions_of_corruption,
valid.freedom_to_make_life_choices, 1)
p = np.poly1d(z)

plt.plot(valid.perceptions_of_corruption,
p(valid.perceptions_of_corruption),linestyle = "dotted")
```

```
plt.annotate(f"p(x) = {p}", (0.2,0.7))
plt.annotate(f"R =
{l1.freedom_to_make_life_choices.perceptions_of_corruption:.3f}",
(0.2,0.68))

plt.show()
```



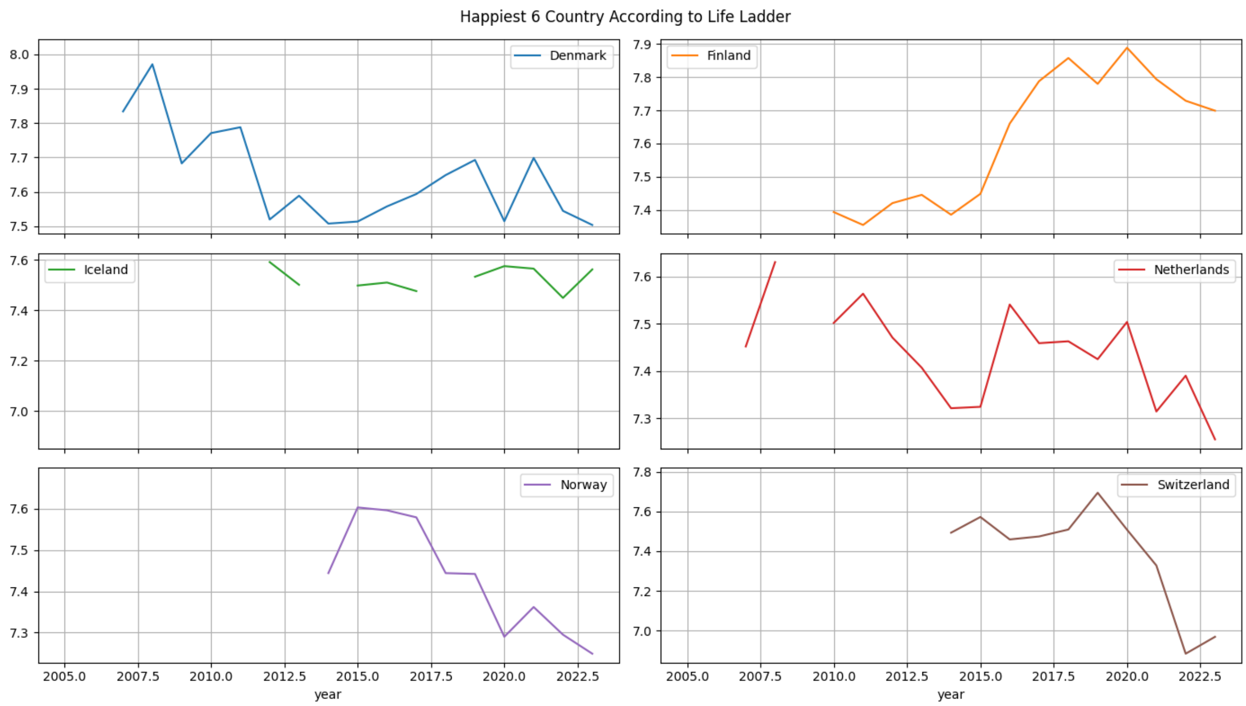

```
valid = modded_df[["generosity", "life_ladder"]]

plt.figure(figsize=(10, 8))
plt.scatter(valid.generosity, valid.life_ladder, color="#A41D70",
edgecolor="black", alpha=0.8)
plt.ylabel("Life ladder")
plt.xlabel("Generosity")
plt.title("Generosity vs Life ladder (weak correlation)")

z = np.polyfit(valid.generosity, valid.life_ladder, 1)
p = np.poly1d(z)
```

```
plt.plot(valid.generosity, p(valid.generosity),linestyle = "dotted")
plt.annotate(f"p(x) = {p}", (0.41,6.9))
plt.annotate(f"R = {l1.life_ladder.generosity:.3f}", (0.41,6.77))

plt.show()
```

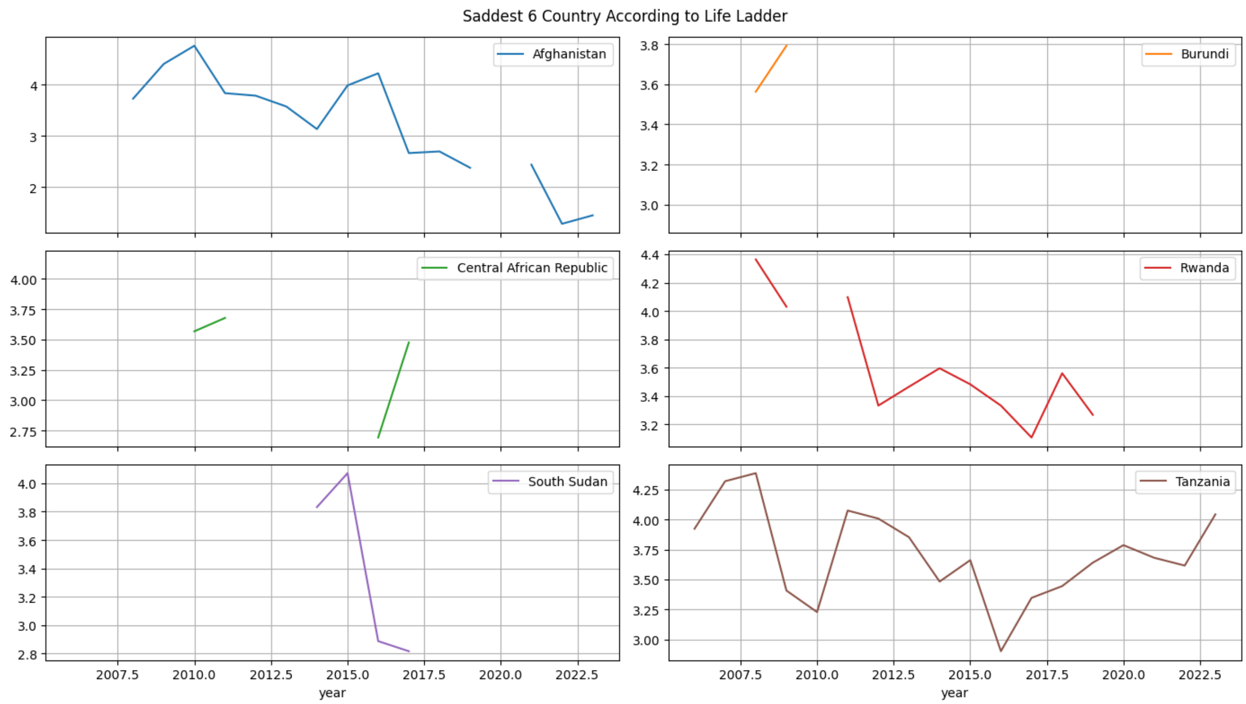



```
top_countries = df.groupby("country_name")
["life_ladder"].mean().nlargest(6).index

df_filtered = df[df.country_name.isin(top_countries)]

df_pivot = df_filtered.pivot(index="year", columns="country_name",
values="life_ladder")

df_pivot.plot(subplots=True, layout=(3, 2), figsize=(14, 8),
title="Happiest 6 Country According to Life Ladder", sharex=True,
grid=True)
```

```
plt.tight_layout()
plt.show()
```




```
top_countries = df.groupby("country_name")
["life_ladder"].mean().nsmallest(6).index

df_filtered = df[df.country_name.isin(top_countries)]

df_pivot = df_filtered.pivot(index="year", columns="country_name",
values="life_ladder")

df_pivot.plot(subplots=True, layout=(3, 2), figsize=(14, 8),
title="Saddest 6 Country According to Life Ladder", sharex=True,
grid=True)

plt.tight_layout()
plt.show()
```



- The number of available data points for the countries analyzed is relatively small. Missing data for some years in the dataset may prevent the trend lines in the graphs from accurately reflecting the actual annual changes.
- It should be noted that in time series created with a small number of observations(last graphs), a single outlier can dramatically affect the direction of the trend, and possible changes in years may remain invisible on the graph (interpolation effect). Therefore, the trends shown should be considered as a general idea rather than a trajectory.
- Missing data points (like Central African Republic graph) might distort the overall graph.