Life Expectency Prediction

import necessary libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
import warnings
warnings.filterwarnings('ignore')
```

Variable Descriptions

Format: variable (type) - description

- country (Nominal) - the country in which the indicators are from (i.e. United States of America or Congo)
- year (Ordinal) - the calendar year the indicators are from (ranging from 2000 to 2015)
- status (Nominal) - whether a country is considered to be 'Developing' or 'Developed' by WHO standards
- life_expectancy (Ratio) - the life expectancy of people in years for a particular country and year
- adult_mortality (Ratio) - the adult mortality rate per 1000 population (i.e. number of people dying between 15 and 60 years per 1000 population); if the rate is 263 then that means 263 people will die out of 1000 between the ages of 15 and 60; another way to think of this is that the chance an individual will die between 15 and 60 is 26.3%
- infant_deaths (Ratio) - number of infant deaths per 1000 population; similar to above, but for infants
- alcohol (Ratio) - a country's alcohol consumption rate measured as liters of pure alcohol consumption per capita
- percentage_expenditure (Ratio) - expenditure on health as a percentage of Gross Domestic Product (gdp)
- hepatitis_b (Ratio) - number of 1 year olds with Hepatitis B immunization over all 1 year olds in population
- measles (Ratio) - number of reported Measles cases per 1000 population

- bmi (Interval/Ordinal) - average Body Mass Index (BMI) of a country's total population
- under-five_deaths (Ratio) - number of people under the age of five deaths per 1000 population
- polio (Ratio) - number of 1 year olds with Polio immunization over the number of all 1 year olds in population
- total_expenditure (Ratio) - government expenditure on health as a percentage of total government expenditure
- diphtheria (Ratio) - Diphtheria tetanus toxoid and pertussis (DTP3) immunization rate of 1 year olds
- hiv/aids (Ratio) - deaths per 1000 live births caused by HIV/AIDS for people under 5; number of people under 5 who die due to HIV/AIDS per 1000 births
- gdp (Ratio) - Gross Domestic Product per capita
- population (Ratio) - population of a country
- thinness_1-19_years (Ratio) - rate of thinness among people aged 10-19 (Note: variable should be renamed to thinness_10-19_years to more accurately represent the variable)
- thinness_5-9_years (Ratio) - rate of thinness among people aged 5-9
- income_composition_of_resources (Ratio) - Human Development Index in terms of income composition of resources (index ranging from 0 to 1) schooling (Ratio) - average number of years of schooling of a population - ICOR measures how good a country is at utilizing its resources.

setting the style of the notebook to be monokai theme

```
from jupyterthemes import jtplot
jtplot.style(theme = 'oceans16', context = 'notebook', ticks = True,
grid = False)
```

This line of code is important to ensure that we are able to see the x and y axes clearly If you don't run this code line, you will notice that the xlabel and ylabel on any plot is black on black and it will be hard to see them.

Perform Exploratory Data Analysis and Visualization

Read the csv file

```
life_expectancy_df = pd.read_csv('Life Expectancy Data.csv')
```

Remove spaces from column names

```
life_expectancy_df.columns = life_expectancy_df.columns.str.strip()
```

Display 5 top rows

```
life_expectancy_df.head()
```

```
       Country  Year      Status  Life expectancy  Adult Mortality  \
0  Afghanistan  2015  Developing             65.0            263.0
1  Afghanistan  2014  Developing             59.9            271.0
2  Afghanistan  2013  Developing             59.9            268.0
3  Afghanistan  2012  Developing             59.5            272.0
4  Afghanistan  2011  Developing             59.2            275.0

   infant deaths  Alcohol  percentage expenditure  Hepatitis B
Measles  ...  \
0             62     0.01               71.279624         65.0
1154  ...
1             64     0.01               73.523582         62.0
492  ...
2             66     0.01               73.219243         64.0
430  ...
3             69     0.01               78.184215         67.0
2787  ...
4             71     0.01                7.097109         68.0
3013  ...

   Polio  Total expenditure  Diphtheria  HIV/AIDS         GDP
Population  \
0    6.0               8.16        65.0       0.1  584.259210
33736494.0
1   58.0               8.18        62.0       0.1  612.696514
327582.0
2   62.0               8.13        64.0       0.1  631.744976
31731688.0
3   67.0               8.52        67.0       0.1  669.959000
3696958.0
4   68.0               7.87        68.0       0.1   63.537231
2978599.0

   thinness  1-19 years  thinness 5-9 years  Income composition of
resources  \
0                  17.2                17.3
0.479
1                  17.5                17.5
0.476
2                  17.7                17.7
0.470
3                  17.9                18.0
0.463
```

```
4                   18.2                 18.2
0.454

   Schooling
0       10.1
1       10.0
2        9.9
3        9.8
4        9.5

[5 rows x 22 columns]
```

Display 5 borttom rows

```
life_expectancy_df.tail()

       Country  Year      Status  Life expectancy  Adult Mortality  \
2933  Zimbabwe  2004  Developing             44.3            723.0
2934  Zimbabwe  2003  Developing             44.5            715.0
2935  Zimbabwe  2002  Developing             44.8             73.0
2936  Zimbabwe  2001  Developing             45.3            686.0
2937  Zimbabwe  2000  Developing             46.0            665.0

      infant deaths  Alcohol  percentage expenditure  Hepatitis B
Measles  \
2933             27     4.36                    0.0         68.0
31
2934             26     4.06                    0.0          7.0
998
2935             25     4.43                    0.0         73.0
304
2936             25     1.72                    0.0         76.0
529
2937             24     1.68                    0.0         79.0
1483

      ...  Polio  Total expenditure  Diphtheria  HIV/AIDS         GDP
\
2933  ...   67.0               7.13        65.0      33.6  454.366654

2934  ...    7.0               6.52        68.0      36.7  453.351155

2935  ...   73.0               6.53        71.0      39.8   57.348340

2936  ...   76.0               6.16        75.0      42.1  548.587312

2937  ...   78.0               7.10        78.0      43.5  547.358878


      Population  thinness  1-19 years  thinness 5-9 years  \
2933  12777511.0                   9.4                 9.4
```

```
2934  12633897.0                    9.8                  9.9
2935    125525.0                    1.2                  1.3
2936  12366165.0                    1.6                  1.7
2937  12222251.0                   11.0                 11.2

      Income composition of resources  Schooling
2933                            0.407        9.2
2934                            0.418        9.5
2935                            0.427       10.0
2936                            0.427        9.8
2937                            0.434        9.8

[5 rows x 22 columns]
```

Display summary statistics

```
life_expectancy_df.describe()

              Year  Life expectancy  Adult Mortality  infant deaths  \
count  2938.000000      2928.000000      2928.000000    2938.000000
mean   2007.518720        69.224932       164.796448      30.303948
std       4.613841         9.523867       124.292079     117.926501
min    2000.000000        36.300000         1.000000       0.000000
25%    2004.000000        63.100000        74.000000       0.000000
50%    2008.000000        72.100000       144.000000       3.000000
75%    2012.000000        75.700000       228.000000      22.000000
max    2015.000000        89.000000       723.000000    1800.000000

           Alcohol  percentage expenditure  Hepatitis B        Measles
\
count  2744.000000             2938.000000  2385.000000    2938.000000

mean      4.602861              738.251295    80.940461    2419.592240

std       4.052413             1987.914858    25.070016   11467.272489

min       0.010000                0.000000     1.000000       0.000000

25%       0.877500                4.685343    77.000000       0.000000

50%       3.755000               64.912906    92.000000      17.000000

75%       7.702500              441.534144    97.000000     360.250000

max      17.870000            19479.911610    99.000000  212183.000000


               BMI  under-five deaths        Polio  Total expenditure
\
count  2904.000000        2938.000000  2919.000000         2712.00000
```

```
mean     38.321247          42.035739    82.550188             5.93819

std      20.044034         160.445548    23.428046             2.49832

min       1.000000           0.000000     3.000000             0.37000

25%      19.300000           0.000000    78.000000             4.26000

50%      43.500000           4.000000    93.000000             5.75500

75%      56.200000          28.000000    97.000000             7.49250

max      87.300000        2500.000000    99.000000            17.60000


        Diphtheria     HIV/AIDS            GDP    Population  \
count  2919.000000  2938.000000    2490.000000  2.286000e+03
mean     82.324084     1.742103    7483.158469  1.275338e+07
std      23.716912     5.077785   14270.169342  6.101210e+07
min       2.000000     0.100000       1.681350  3.400000e+01
25%      78.000000     0.100000     463.935626  1.957932e+05
50%      93.000000     0.100000    1766.947595  1.386542e+06
75%      97.000000     0.800000    5910.806335  7.420359e+06
max      99.000000    50.600000  119172.741800  1.293859e+09

       thinness  1-19 years  thinness 5-9 years  \
count           2904.000000         2904.000000
mean               4.839704            4.870317
std                4.420195            4.508882
min                0.100000            0.100000
25%                1.600000            1.500000
50%                3.300000            3.300000
75%                7.200000            7.200000
max               27.700000           28.600000

       Income composition of resources    Schooling
count                      2771.000000  2775.000000
mean                          0.627551    11.992793
std                           0.210904     3.358920
min                           0.000000     0.000000
25%                           0.493000    10.100000
50%                           0.677000    12.300000
75%                           0.779000    14.300000
max                           0.948000    20.700000
```

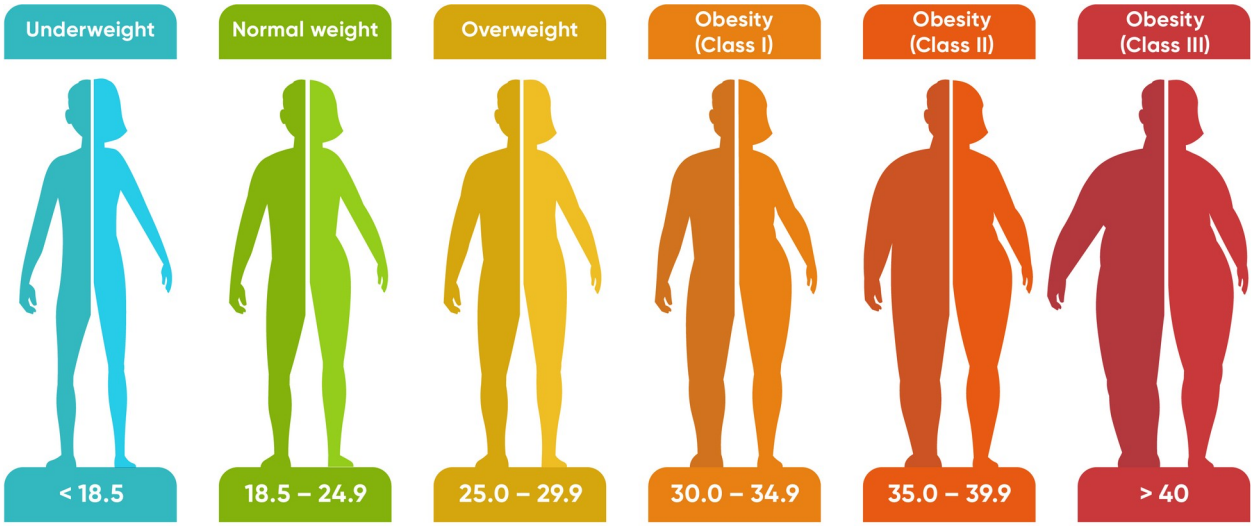
Comprehensive Analysis of Dataset Variables

BMI interpretaion

- Mean BMI of 38.32 indicates a tendency towards "Obesity (Class II)" on average.
- Wide spread, as indicated by standard deviation, reflects variability in BMI values.

- Minimum BMI of 1 suggests a potential error or outlier, requiring further investigation.
- Maximum BMI of 87.3 also raises concerns as an extreme outlier, necessitating careful validation.
- Addressing outliers is crucial for accurate interpretation and understanding the dataset's overall distribution.

```
from IPython.display import Image, display

# Specify the file path of your image
image_path = 'BMI.png'

# Display the image
display(Image(filename=image_path))
```



Adult Mortality interpretaion

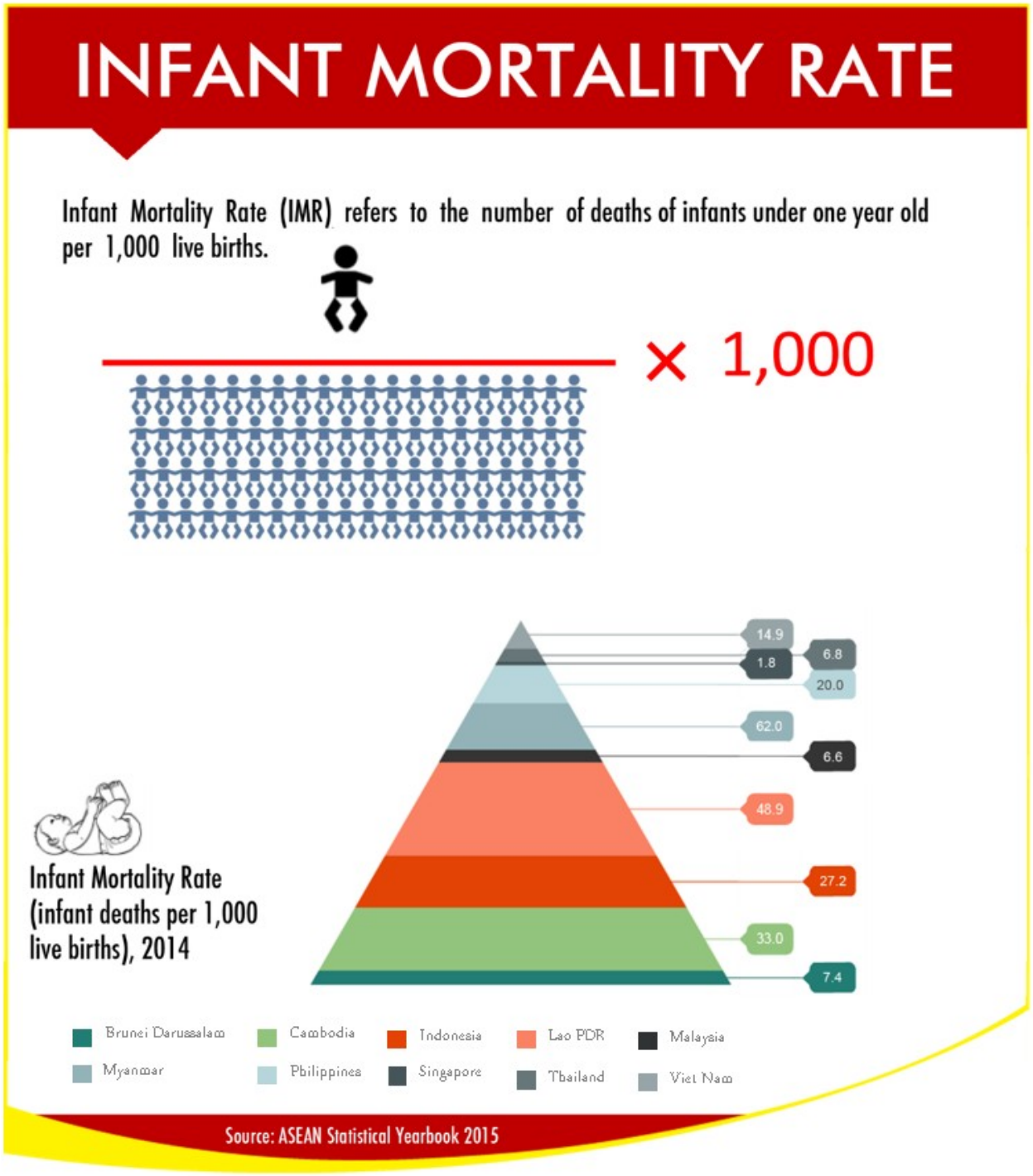
- Mean adult mortality rate of 164.79 suggests a moderate average risk of dying before 60.
- High variability, with a standard deviation of 124.29, indicates diverse mortality rates
- Minimum rate of 1 may be an error, while the maximum of 723 needs careful validation as an extre value.
- The dataset et aligns with WHO's definition of adult mortality, emphasizing irelevance.
- Scrutinizing outliers is essential for precise interpretation and a comprehensive view of mortality patterns.rns.

Infant deaths interpretaion

- Mean infant deaths ratio of 30.30 implies an average of 30.30 infant deaths per 1000 population.
- High variability, with a standard deviation of 117.92, indicates diverse infant mortality ratios.
- The minimum ratio of 0 suggests that some areas or observations reported no infant deaths.
- The maximum ratio of 1800 raises concerns as an extreme outlier, necessitating careful validation.

- Scrutinizing outliers is crucial for accurate interpretation and understanding the distribution of infant mortality ratios.ios.atios.atios.

```
from IPython.display import Image, display

# Specify the file path of your image
image_path = 'Infant-Mortality-Rate.png'

# Display the image
display(Image(filename=image_path))
```

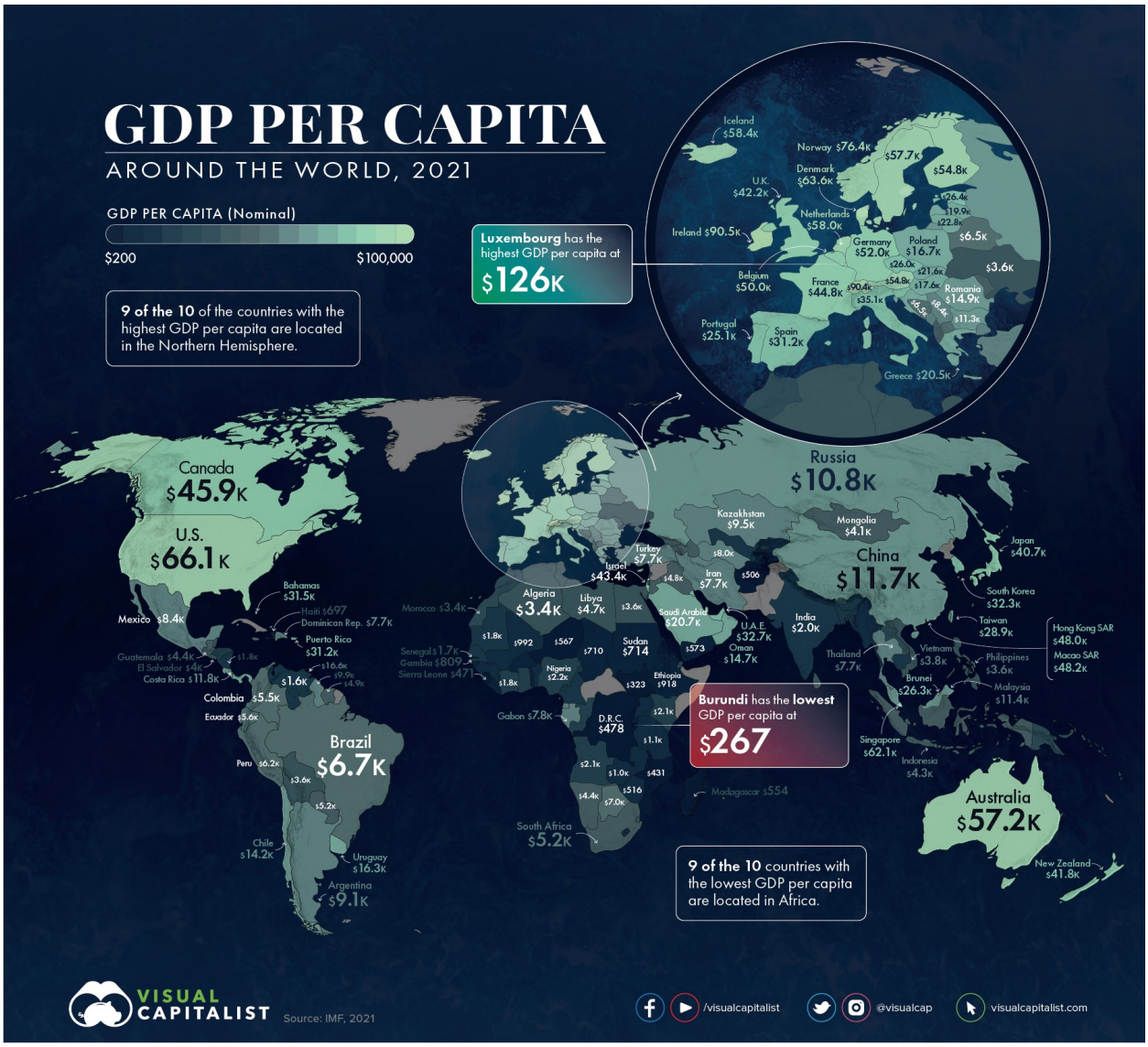


GDP interpretaion

- The maximum GDP of 119172 indicates potential outliers, requiring validation.
- The minimum GDP of 1.68 may suggest anomalies or data errors, prompting further investigation.

```
from IPython.display import Image, display
```

```
# Specify the file path of your image
image_path = 'map-gdp-per-capita-large.jpg'

# Display the image
display(Image(filename=image_path))
```



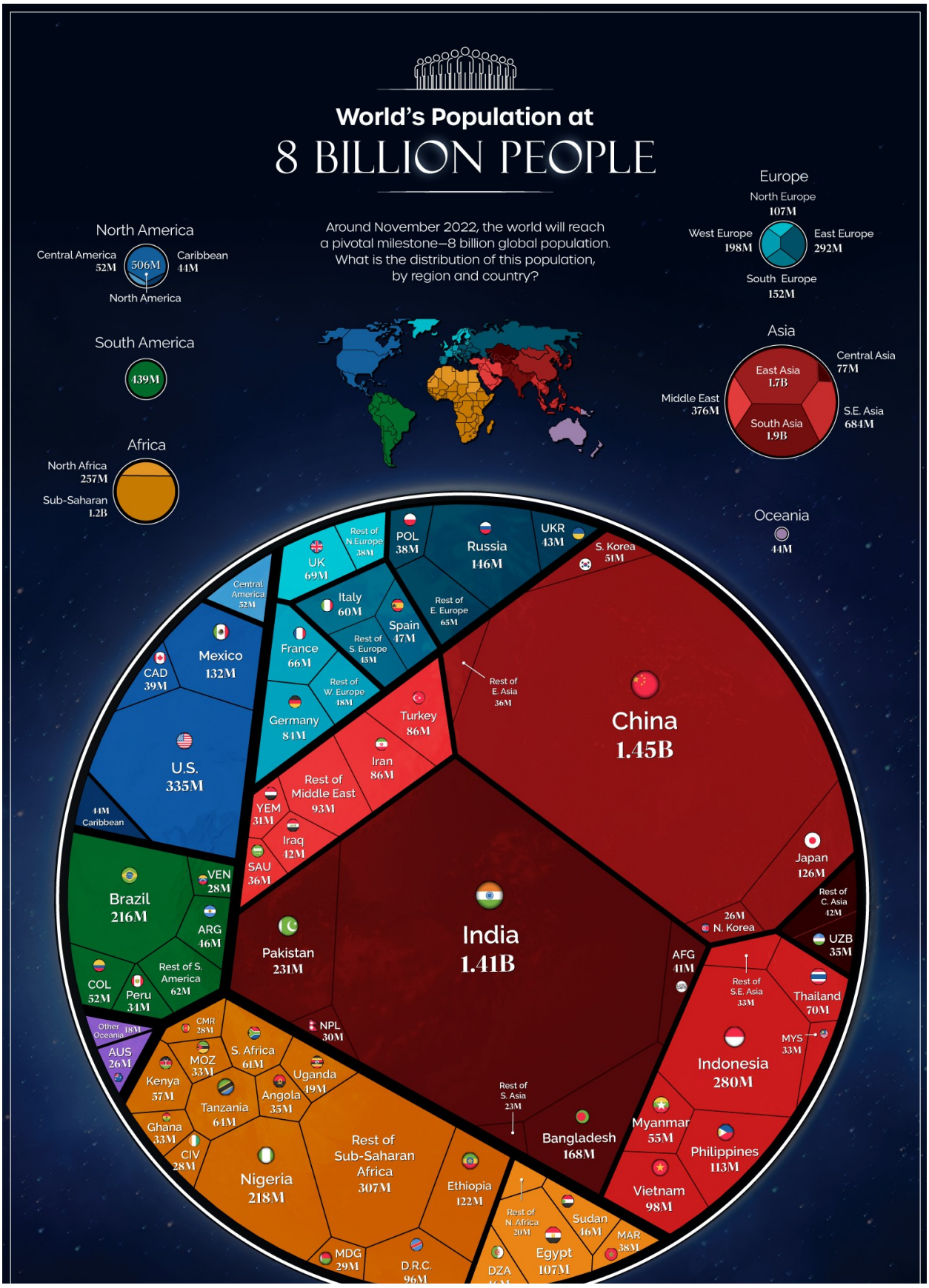
COLLABORATORS RESEARCH + WRITING Avery Koop, Raul Amoros | DESIGN + ART DIRECTION Amy Kuo, Christina Kostandi, Melissa Haavisto

Population interpretaion

- The dataset's population statistics, spanning from a minimum of 34 to a maximum of 1.29 billion, highlight substantial variability in demographic profiles.
- Despite the mean population of 12.7 million indicating a moderate average, the high standard deviation (61 million) signifies significant diversity among countries or regions.

- Considering both the extreme maximum value of 1.29 billion and the low minimum value of 34 as potential outliers, careful validation is crucial to ensure data accuracy and identify anomalies in the dataset.

```
from IPython.display import Image, display

# Specify the file path of your image
image_path = 'world-population-at-8-billion.jpg'

# Display the image
display(Image(filename=image_path))
```


World's Population at
8 BILLION PEOPLE
Around November 2022, the world will reach a pivotal milestone—8 billion global population. What is the distribution of this population, by region and country?
North America
Central America 52M
Caribbean 44M
506M
North America
South America
439M
Africa
North Africa 257M
Sub-Saharan 1.2B
Europe
North Europe 107M
West Europe 198M
East Europe 292M
South Europe 152M
Asia
Central Asia 77M
East Asia 1.7B
Middle East 376M
South Asia 1.9B
S.E. Asia 684M
Oceania
44M
China 1.45B
India 1.41B
U.S. 335M
Brazil 216M
Russia 146M
Indonesia 280M
Pakistan 231M
Nigeria 218M
Japan 126M
Mexico 132M
Bangladesh 168M
Ethiopia 122M
Philippines 113M
Egypt 107M
Vietnam 98M
Turkey 86M
Iran 86M
Germany 84M
Thailand 70M
France 66M
UK 69M
Italy 60M
Spain 47M
Myanmar 55M
Kenya 57M
Tanzania 64M
S. Africa 61M
Colombia 52M
Peru 34M
Argentina 46M
Venezuela 28M
Canada 39M
Poland 38M
Ukraine 43M
S. Korea 51M
Rest of N. Europe 38M
Rest of E. Europe 65M
Rest of S. Europe 45M
Rest of W. Europe 48M
Central America 52M
Rest of E. Asia 36M
Rest of Middle East 93M
Yemen 31M
Iraq 42M
Saudi Arabia 36M
Afghanistan 41M
26M N. Korea
Rest of C. Asia 42M
Uzbekistan 35M
Rest of S.E. Asia 33M
Malaysia 33M
Nepal 30M
Rest of S. Asia 23M
Rest of Sub-Saharan Africa 307M
Sudan 46M
Morocco 38M
Algeria 43M
Rest of N. Africa 20M
Uganda 49M
Angola 35M
Mozambique 33M
Cameroon 28M
Ghana 33M
Ivory Coast 28M
Madagascar 29M
D.R.C. 96M
Rest of S. America 62M
Caribbean 44M
Other Oceania 18M
Australia 26M

Display Index, Column names and Non-null Count

```
info_data = []
for col in life_expectancy_df.columns:
    dtype = life_expectancy_df[col].dtype
    non_null_count = life_expectancy_df[col].count()
    info_data.append([col, dtype, non_null_count])

info_df = pd.DataFrame(info_data, columns=['Column', 'Data Type',
'Non-null Count'])
display(info_df)
```

	Column	Data Type	Non-null Count
0	Country	object	2938
1	Year	int64	2938
2	Status	object	2938
3	Life expectancy	float64	2928
4	Adult Mortality	float64	2928
5	infant deaths	int64	2938
6	Alcohol	float64	2744
7	percentage expenditure	float64	2938
8	Hepatitis B	float64	2385
9	Measles	int64	2938
10	BMI	float64	2904
11	under-five deaths	int64	2938
12	Polio	float64	2919
13	Total expenditure	float64	2712
14	Diphtheria	float64	2919
15	HIV/AIDS	float64	2938
16	GDP	float64	2490
17	Population	float64	2286
18	thinness 1-19 years	float64	2904
19	thinness 5-9 years	float64	2904
20	Income composition of resources	float64	2771
21	Schooling	float64	2775

Checking for missing values

```
null_counts = life_expectancy_df.isnull().sum()
null_counts_df = pd.DataFrame(null_counts, columns=['Null Count'])
display(null_counts_df)
```

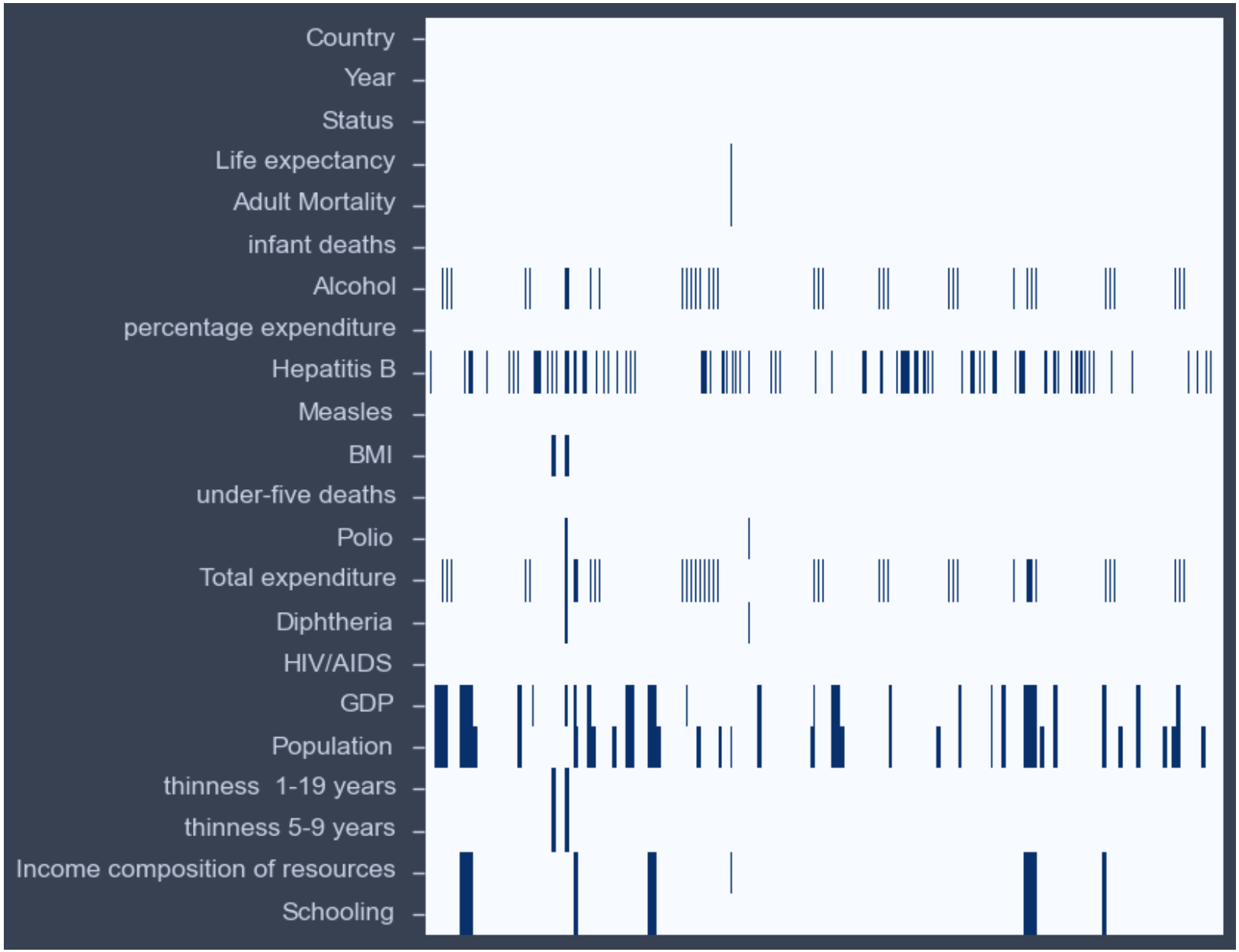
	Null Count
Country	0
Year	0
Status	0
Life expectancy	10
Adult Mortality	10
infant deaths	0
Alcohol	194
percentage expenditure	0

```
Hepatitis B                         553
Measles                               0
BMI                                  34
under-five deaths                     0
Polio                                19
Total expenditure                   226
Diphtheria                           19
HIV/AIDS                              0
GDP                                 448
Population                          652
thinness  1-19 years                 34
thinness 5-9 years                   34
Income composition of resources     167
Schooling                           163
```

Show missing values in each column using **heatmap**

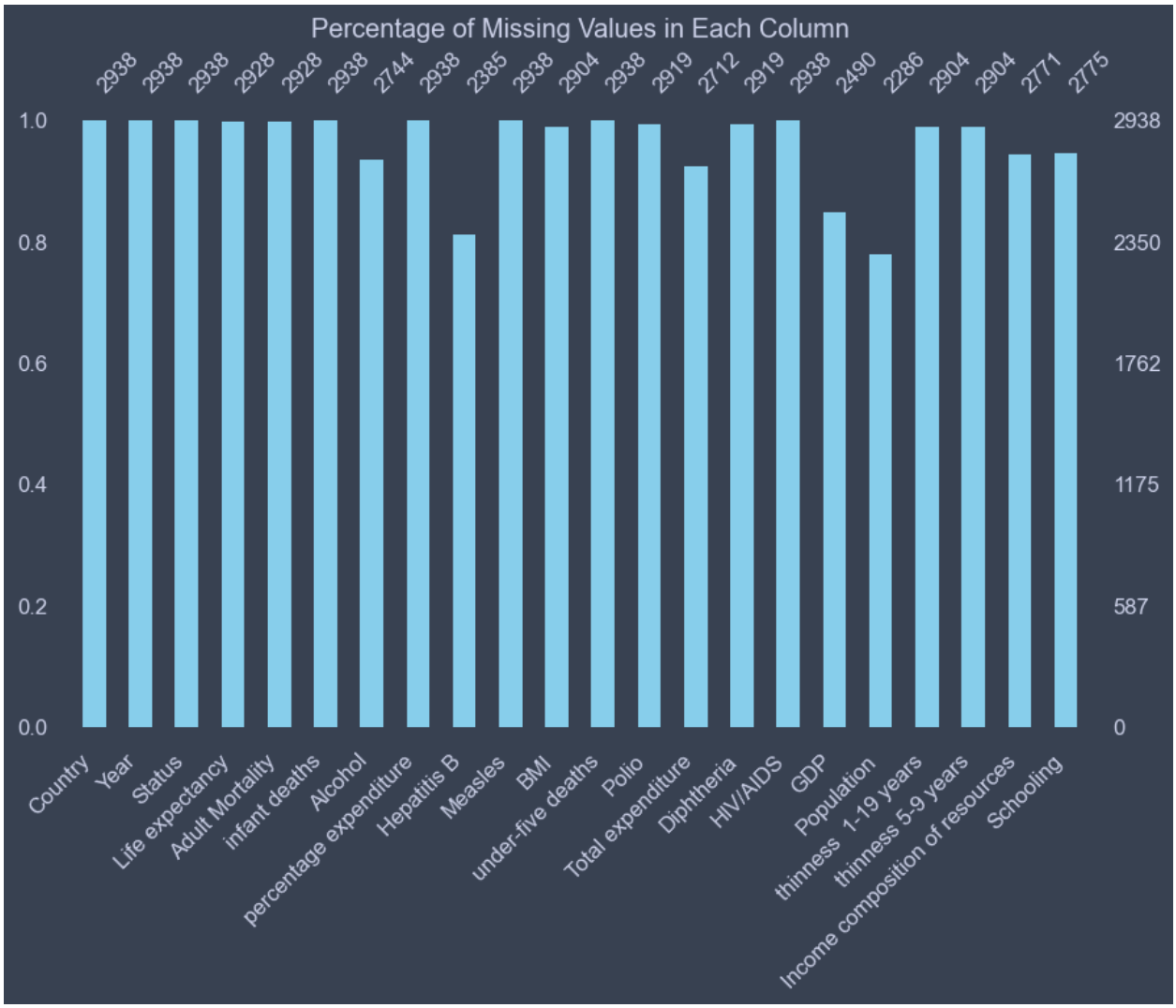
```
# check if there are any Null values
sns.heatmap(life_expectancy_df.isnull(), yticklabels = False, cbar =
False, cmap="Blues")
```

```
<Axes: >
```


Country
Year
Status
Life expectancy
Adult Mortality
infant deaths
Alcohol
percentage expenditure
Hepatitis B
Measles
BMI
under-five deaths
Polio
Total expenditure
Diphtheria
HIV/AIDS
GDP
Population
thinness 1-19 years
thinness 5-9 years
Income composition of resources
Schooling

Plot a bar chart of missing values (All columns)

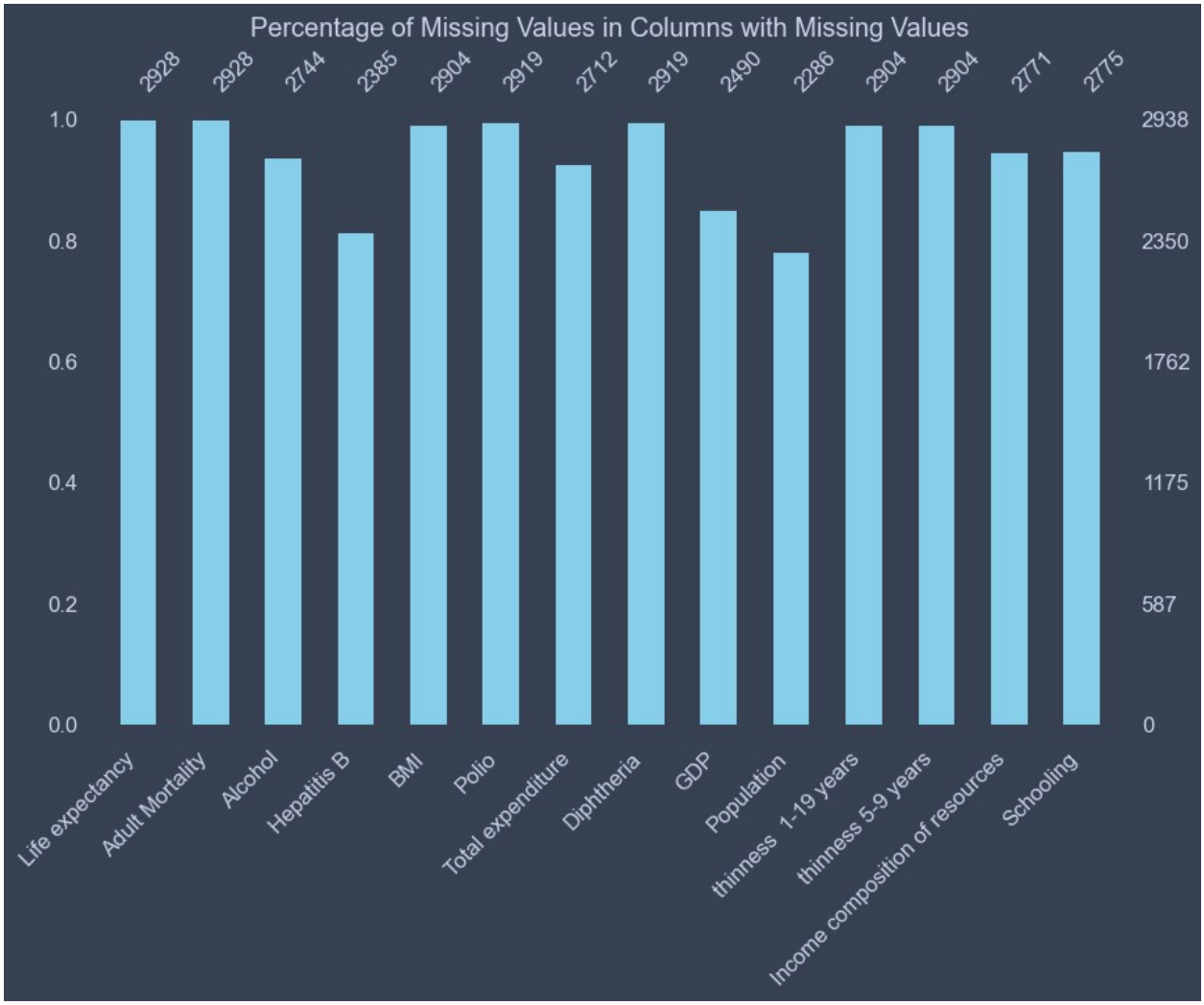
```
import missingno as msno

msno.bar(life_expectancy_df, figsize=(10, 6), color='skyblue',
fontsize=12)
plt.title("Percentage of Missing Values in Each Column")
plt.show()
```



Plot a bar chart of missing values (Columns with missing values)

```
# Filter columns with missing values
columns_with_missing_values =
life_expectancy_df.columns[life_expectancy_df.isnull().any()]

# Create a bar chart of missing values for selected columns
msno.bar(life_expectancy_df[columns_with_missing_values], figsize=(10,
6), color='skyblue', fontsize=12)
plt.title("Percentage of Missing Values in Columns with Missing
```

```
Values")
plt.show()
```

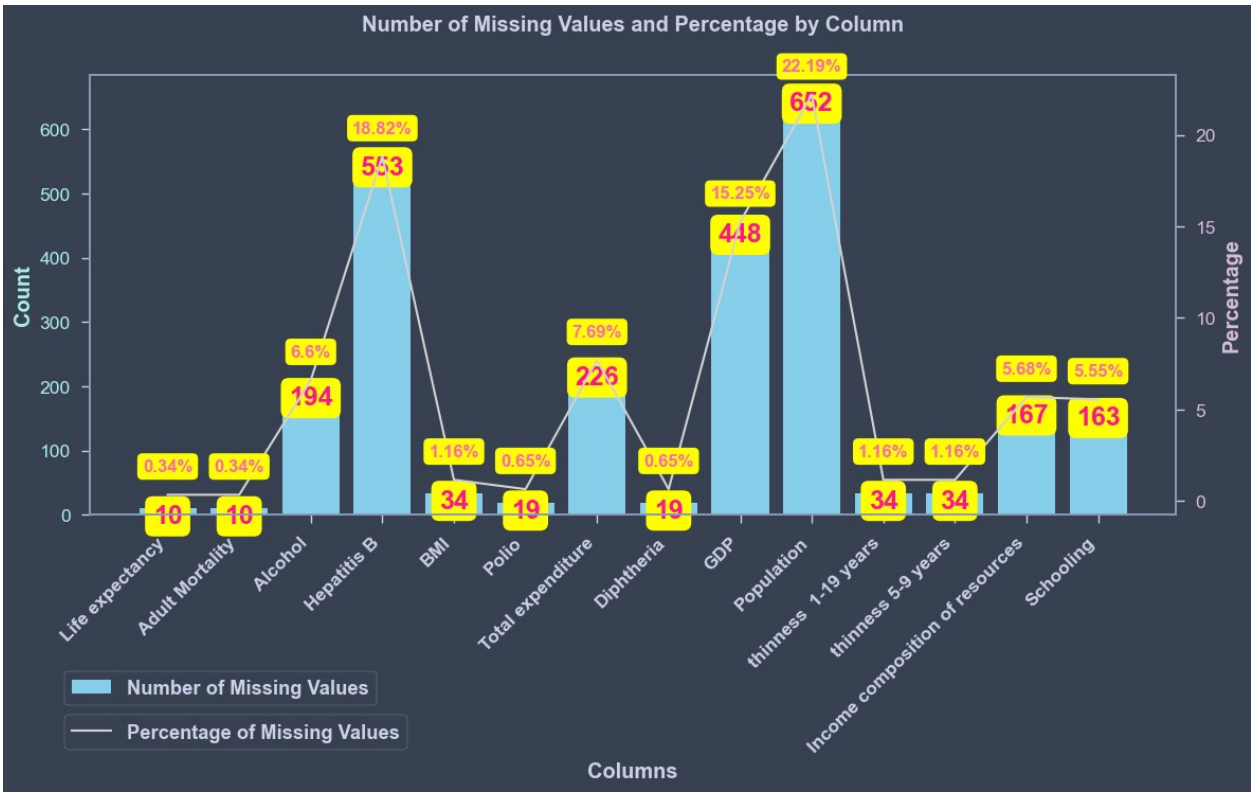


Number and Percentage of Missing values by column

```
from matplotlib.font_manager import FontProperties


missing_values = life_expectancy_df.isnull().sum()
missing_values = missing_values[missing_values != 0]

# Calculate the percentage of missing values
missing_percentage = (missing_values / len(life_expectancy_df)) * 100

# Create a bar chart for the count of missing values
fig, ax1 = plt.subplots(figsize=(12, 9))

# Bar for the number of missing values
bar1 = ax1.bar(missing_values.index, missing_values, color='skyblue',
```

```
label='Number of Missing Values')

ax1.set_xlabel('Columns', fontweight='bold')  # Set fontweight for x-
axis label
ax1.set_ylabel('Count', color='paleturquoise', fontweight='bold')
ax1.tick_params(axis='y', labelcolor='paleturquoise')
ax1.set_title('Number of Missing Values and Percentage by Column',
pad=30, fontweight='bold')  # Adjust pad for space

# Display the values on top of the bars with adjusted vertical
positioning
for bar in bar1:
    yval = bar.get_height()
    # Add a background color to the text
    ax1.text(bar.get_x() + bar.get_width()/2, yval + 0.01 *
max(missing_values), round(yval, 2),
             ha='center', va='top', color='deeppink', fontsize=18,
fontweight='bold', bbox=dict(facecolor='yellow', edgecolor='none',
boxstyle='round'))

# Create a second y-axis for the percentage of missing values using a
line chart
ax2 = ax1.twinx()
line2 = ax2.plot(missing_percentage.index, missing_percentage,
linestyle='-', color='lightgray', label='Percentage of Missing
Values')


ax2.set_ylabel('Percentage', color='thistle', fontweight='bold')
ax2.tick_params(axis='y', labelcolor='thistle')

# Display the values on top of the line chart with adjusted vertical
positioning
for i, value in enumerate(missing_percentage):
    # Add a background color to the text
    ax2.text(missing_percentage.index[i], value + 0.05 *
max(missing_percentage), f'{round(value, 2)}%',
             ha='center', va='bottom', color='hotpink', fontsize=12,
fontweight='bold', bbox=dict(facecolor='yellow', edgecolor='none',
boxstyle='round'))

# Set x-axis ticks and labels
ax1.set_xticks(np.arange(len(missing_values)))
ax1.set_xticklabels(missing_values.index, rotation=45, ha='right',
fontweight='bold')

# Display both legends outside the chart below the x-axis with some
space
legend_font = FontProperties(weight='bold')  # Create FontProperties
object for legend
```

```
ax1_legend = ax1.legend(loc='lower left', bbox_to_anchor=(-0.05, -
0.5), borderaxespad=1.5, prop=legend_font)
ax2_legend = ax2.legend(loc='lower left', bbox_to_anchor=(-0.05, -
0.6), borderaxespad=1.5, prop=legend_font)

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



Dealing with Missing Values

It looks like there are a lot of columns containing null values, since this is time series data assorted by country, the best course of action would be to interpolate the data by country. However, when attempting to interpolate by country it doesn't fill in any values as the countries' data for all the null values are null for each year, therefore imputation by year may be the best possible method here. Imputation of each year's mean is done below.

```
imputed_data = []

for year in life_expectancy_df['Year'].unique():
    year_data = life_expectancy_df[life_expectancy_df['Year'] ==
year].copy()

    for col in list(year_data.columns)[3:]:
        year_data[col] =
year_data[col].fillna(year_data[col].dropna().mean()).copy()
```

```
    imputed_data.append(year_data)

life_expectancy_df = pd.concat(imputed_data).copy()

# Verifying null-values after applying above methods.
null_counts = pd.DataFrame(life_expectancy_df.isnull().sum(),
columns=['Null Count'])
display(null_counts)
```

	Null Count
Country	0
Year	0
Status	0
Life expectancy	0
Adult Mortality	0
infant deaths	0
Alcohol	0
percentage expenditure	0
Hepatitis B	0
Measles	0
BMI	0
under-five deaths	0
Polio	0
Total expenditure	0
Diphtheria	0
HIV/AIDS	0
GDP	0
Population	0
thinness 1-19 years	0
thinness 5-9 years	0
Income composition of resources	0
Schooling	0

Checking for Ouliers

```
# Create a dictionary of columns.
col_dict = {'Life expectancy': 1, 'Adult Mortality': 2,
            'infant deaths': 3, 'Alcohol': 4, 'percentage
expenditure': 5, 'Hepatitis B': 6,
            'Measles': 7, 'BMI': 8, 'under-five deaths': 9, 'Polio':
10, 'Total expenditure': 11,
            'Diphtheria': 12, 'HIV/AIDS': 13, 'GDP': 14, 'Population':
15, 'thinness  1-19 years': 16,
            'thinness 5-9 years': 17, 'Income composition of
resources': 18, 'Schooling': 19}

# Create a figure and axis
plt.figure(figsize=(20, 30))
i = 1  # Initialize subplot index
```

```
# Iterate through the columns
for key, value in col_dict.items():
    plt.subplot(5, 4, i)

    # Use Seaborn boxplot with custom color for all elements
    sns.boxplot(x=life_expectancy_df[key], color="skyblue", width=0.5,
linewidth=1.5,
                boxprops=dict(facecolor='skyblue', edgecolor='black'),
                whiskerprops=dict(color='hotpink'),
                capprops=dict(color='hotpink'),
                flierprops=dict(markerfacecolor='turquoise',
markeredgecolor='hotpink'),
                medianprops=dict(color='hotpink'))

    plt.title(key)
    i += 1  # Increment subplot index

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```


Life expectancy
40 50 60 70 80 90
Life expectancy
Adult Mortality
0 100 200 300 400 500 600 700
Adult Mortality
infant deaths
0 250 500 750 1000 1250 1500 1750
infant deaths
Alcohol
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Alcohol
percentage expenditure
0 5000 10000 15000 20000
percentage expenditure
Hepatitis B
0 20 40 60 80 100
Hepatitis B
Measles
0 50000 100000 150000 200000
Measles
BMI
0 20 40 60 80
BMI
under-five deaths
0 500 1000 1500 2000 2500
under-five deaths
Polio
0 20 40 60 80 100
Polio
Total expenditure
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Total expenditure
Diphtheria
0 20 40 60 80 100
Diphtheria
HIV/AIDS
0 10 20 30 40 50
HIV/AIDS
GDP
0 20000 40000 60000 80000 100000 120000
GDP
Population
0.0 0.2 0.4 0.6 0.8 1.0 1.2
1e9
Population
thinness 1-19 years
0 5 10 15 20 25
thinness 1-19 years
thinness 5-9 years
Income composition of resources
Schooling

Tukey's method

Tukey's method, also known as the Tukey's fences or Tukey's range test, is a statistical technique designed for identifying outliers in a dataset. Developed by the eminent statistician John Tukey, this method relies on the interquartile range (IQR), which is the range between the first quartile (Q1) and the third quartile (Q3) of a distribution. Tukey proposed defining a "fence" around the IQR, with outliers identified as values falling outside these fences. Typically, the lower fence is set at:

$$Q1 - k \times \text{IQR}$$

and the upper fence is set a:

$$Q3 + k \times \text{IQR}$$

where (k) is a constant multiplier. A common choice for (k) is 1.5, although other values may be used depending on the desired sensitivity to outliers. Tukey's method is widely employed for its simplicity and effectiveness in detecting outliers while accounting for the variability in the central portion of the distributio

Calculate number of outliers and its percentage in each variable using Tukey's method

```
outliers_data = []

for variable in col_dict.keys():
    q75, q25 = np.percentile(life_expectancy_df[variable], [75, 25])
    iqr = q75 - q25

    min_val = q25 - (iqr * 1.5)
    max_val = q75 + (iqr * 1.5)

    outliers_count = len((np.where((life_expectancy_df[variable] >
max_val) | (life_expectancy_df[variable] < min_val))[0]))
    outliers_percentage = outliers_count * 100 /
len(life_expectancy_df)

    outliers_data.append({
        'Variable': variable,
        'Outliers Count': outliers_count,
        'Outliers Percentage': outliers_percentage
    })

outliers_table = pd.DataFrame(outliers_data)
display(outliers_table)
```

```
                          Variable  Outliers Count  Outliers
Percentage
0                 Life expectancy              17
0.578625
1                 Adult Mortality              86
2.927161
```

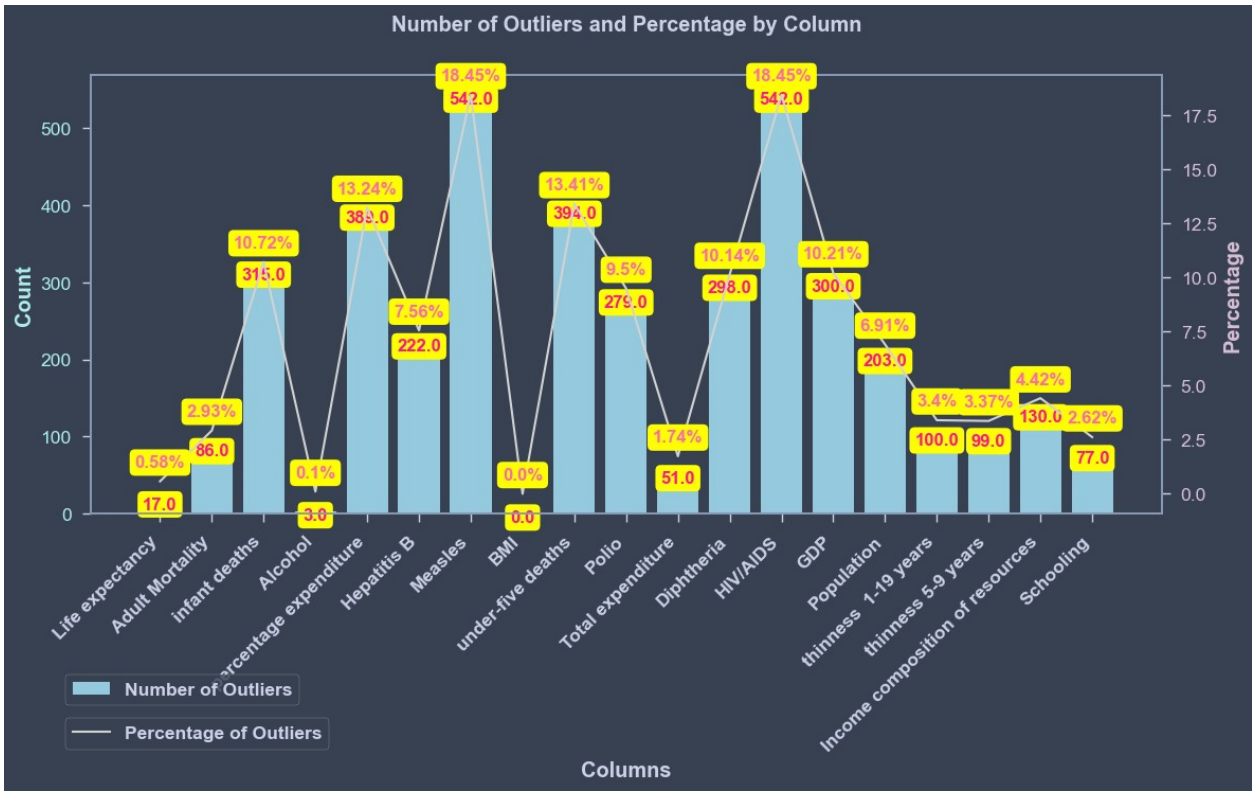
```
2                      infant deaths              315
10.721579
3                            Alcohol                3
0.102110
4            percentage expenditure              389
13.240300
5                        Hepatitis B              222
7.556161
6                            Measles              542
18.447924
7                                BMI                0
0.000000
8                  under-five deaths              394
13.410483
9                               Polio              279
9.496256
10                 Total expenditure               51
1.735875
11                       Diphtheria              298
10.142954
12                         HIV/AIDS              542
18.447924
13                               GDP              300
10.211028
14                        Population              203
6.909462
15             thinness  1-19 years              100
3.403676
16                thinness 5-9 years               99
3.369639
17  Income composition of resources              130
4.424779
18                          Schooling               77
2.620830
```

Number and Percentage of ouliers by column

```
# Calculate number of outliers and its percentage in each variable
using Tukey's method
outliers_count = []
outliers_percentage = []

for variable in col_dict.keys():
    q75, q25 = np.percentile(life_expectancy_df[variable], [75, 25])
    iqr = q75 - q25

    min_val = q25 - (iqr * 1.5)
    max_val = q75 + (iqr * 1.5)

    outliers = np.where((life_expectancy_df[variable] > max_val) |
```

```
(life_expectancy_df[variable] < min_val))[0]
    outliers_count.append(len(outliers))
    outliers_percentage.append(len(outliers) * 100 /
len(life_expectancy_df))

# Create a DataFrame for visualization
outliers_df = pd.DataFrame({'Variable': col_dict.keys(), 'Outliers
Count': outliers_count, 'Outliers Percentage': outliers_percentage})

# Create a bar chart for the count of outliers
fig, ax1 = plt.subplots(figsize=(12, 9))

# Bar for the number of outliers
bar1 = sns.barplot(x='Variable', y='Outliers Count', data=outliers_df,
color='skyblue', label='Number of Outliers')

ax1.set_xlabel('Columns', fontweight='bold')
ax1.set_ylabel('Count', color='paleturquoise', fontweight='bold')
ax1.tick_params(axis='y', labelcolor='paleturquoise')
ax1.set_title('Number of Outliers and Percentage by Column', pad=30,
fontweight='bold')

# Display the values on top of the bars with adjusted vertical
positioning
for bar in bar1.patches:
    yval = bar.get_height()
    ax1.text(bar.get_x() + bar.get_width()/2, yval + 0.01 *
max(outliers_count), round(yval, 2),
             ha='center', va='top', color='deeppink', fontsize=12,
fontweight='bold', bbox=dict(facecolor='yellow', edgecolor='none',
boxstyle='round'))

# Create a second y-axis for the percentage of outliers using a line
chart
ax2 = ax1.twinx()
line2 = sns.lineplot(x='Variable', y='Outliers Percentage',
data=outliers_df, color='lightgray', label='Percentage of Outliers')

ax2.set_ylabel('Percentage', color='thistle', fontweight='bold')
ax2.tick_params(axis='y', labelcolor='thistle')

# Display the values on top of the line chart with adjusted vertical
positioning
for i, value in enumerate(outliers_percentage):
    ax2.text(outliers_df['Variable'][i], value + 0.5, f'{round(value,
2)}%',
             ha='center', va='bottom', color='hotpink', fontsize=12,
fontweight='bold', bbox=dict(facecolor='yellow', edgecolor='none',
boxstyle='round'))
```

```
# Set x-axis ticks and labels
ax1.set_xticks(np.arange(len(outliers_df)))
ax1.set_xticklabels(outliers_df['Variable'], rotation=45, ha='right',
fontweight='bold')

# Display both legends outside the chart below the x-axis with some
space
legend_font = {'weight': 'bold'}
ax1_legend = ax1.legend(loc='lower left', bbox_to_anchor=(-0.05, -
0.5), borderaxespad=1.5, prop=legend_font)
ax2_legend = ax2.legend(loc='lower left', bbox_to_anchor=(-0.05, -
0.6), borderaxespad=1.5, prop=legend_font)

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



Winsorization: A Robust Approach for Outlier Treatment

Winsorization is a statistical method designed to address outliers in a dataset, offering robustness in the face of extreme values. This technique, akin to Tukey's method, involves capping extreme values by replacing them with values within specified bounds. The process is particularly useful during data preprocessing, ensuring that extreme values do not unduly influence statistical analyses.

Removing Outliers in the variables using Winsorization technique

Link for Winsorization technique:

https://www.investopedia.com/terms/w/winsorized_mean.asp

```
from scipy.stats.mstats import winsorize


# Define winsorization limits for each column
winsorization_limits = {
    'Life expectancy': (0.01, 0),
    'Adult Mortality': (0, 0.03),
    'infant deaths': (0, 0.12),
    'Alcohol': (0, 0.01),
    'percentage expenditure': (0, 0.15),
    'Hepatitis B': (0.11, 0),
    'Measles': (0, 0.19),
    'under-five deaths': (0, 0.15),
    'Polio': (0.095, 0),
    'Total expenditure': (0, 0.02),
    'Diphtheria': (0.20, 0),
    'HIV/AIDS': (0, 0.19),
    'GDP': (0, 0.13),
    'Population': (0, 0.14),
    'thinness  1-19 years': (0, 0.04),
    'thinness 5-9 years': (0, 0.04),
    'Income composition of resources': (0.05, 0),
    'Schooling': (0.025, 0.01)
}


# Create a new DataFrame to store winsorized data
winsorized_df = pd.DataFrame()

# Initialize dictionaries to store the number of outliers and their
percentage for each variable
outliers_count_before = {}
outliers_percentage_before = {}
outliers_count_after = {}
outliers_percentage_after = {}

# Create subplots based on the number of variables
fig, axs = plt.subplots(len(winsorization_limits), 2, figsize=(16, 5 *
len(winsorization_limits)))
axs = axs.flatten()

# Iterate through each column and its winsorization limits
for i, (column, limits) in enumerate(winsorization_limits.items()):
    # Plot original data
    plt.subplot(len(winsorization_limits), 2, 2 * i + 1)
```

```
    sns.boxplot(x=life_expectancy_df[column], color="skyblue",
width=0.5, linewidth=1.5,
                boxprops=dict(facecolor='skyblue', edgecolor='black'),
                whiskerprops=dict(color='hotpink'),
                capprops=dict(color='hotpink'),
                flierprops=dict(markerfacecolor='turquoise',
markeredgecolor='hotpink'),
                medianprops=dict(color='hotpink'))
    plt.title(f"Original {column}")

    # Count the number of outliers in original data using Tukey's
method
    q75, q25 = np.percentile(life_expectancy_df[column], [75, 25])
    iqr = q75 - q25
    min_val = q25 - (iqr * 1.5)
    max_val = q75 + (iqr * 1.5)

    outliers_before = (np.where((life_expectancy_df[column] > max_val)
| (life_expectancy_df[column] < min_val))[0])
    outliers_count_before[column] = len(outliers_before)
    outliers_percentage_before[column] = outliers_count_before[column]
* 100 / len(life_expectancy_df[column])


    # Winsorize the data
    winsorized_data = winsorize(life_expectancy_df[column], limits)

    # Add winsorized data to the new DataFrame
    winsorized_df['winsorized_' + column] = winsorized_data

    # Plot winsorized data
    plt.subplot(len(winsorization_limits), 2, 2 * i + 2)
    sns.boxplot(x=winsorized_data, color="skyblue", width=0.5,
linewidth=1.5,
                boxprops=dict(facecolor='skyblue', edgecolor='black'),
                whiskerprops=dict(color='hotpink'),
                capprops=dict(color='hotpink'),
                flierprops=dict(markerfacecolor='turquoise',
markeredgecolor='hotpink'),
                medianprops=dict(color='hotpink'))
    plt.title(f"Winsorized {column}")

    # Count the number of outliers in winsorized data using Tukey's
method
    q75, q25 = np.percentile(winsorized_data, [75, 25])
    iqr = q75 - q25
    min_val = q25 - (iqr * 1.5)
    max_val = q75 + (iqr * 1.5)

    outliers_after = (np.where((winsorized_data > max_val) |
```

```
(winsorized_data < min_val))[0])
    outliers_count_after[column] = len(outliers_after)
    outliers_percentage_after[column] = outliers_count_after[column] *
100 / len(winsorized_data)




# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```


Original Life expectancy
40
50
60
70
80
90
Life expectancy
Winsorized Life expectancy
50
60
70
80
90
Original Adult Mortality
0
100
200
300
400
500
600
700
Adult Mortality
Winsorized Adult Mortality
0
100
200
300
400
Original infant deaths
0
250
500
750
1000
1250
1500
1750
infant deaths
Winsorized infant deaths
0
10
20
30
40
50
Original Alcohol
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Alcohol
Winsorized Alcohol
0
2
4
6
8
10
12
14
Original percentage expenditure
Winsorized percentage expenditure

Display a table with the number of outliers before and after winsorization

```
from IPython.display import Markdown, display

outliers_table = pd.DataFrame({
    'Variable': list(winsorization_limits.keys()),
    'Outliers Before Winsorization':
list(outliers_count_before.values()),
    'Percentage Before Winsorization':
list(outliers_percentage_before.values()),
    'Outliers After Winsorization':
list(outliers_count_after.values()),
    'Percentage After Winsorization':
list(outliers_percentage_after.values())
})

display(Markdown("\n**Outliers Before and After Winsorization:**"))
display(outliers_table)
```

```
<IPython.core.display.Markdown object>
```

	Variable	Outliers Before Winsorization	\
0	Life expectancy	17	
1	Adult Mortality	86	
2	infant deaths	315	
3	Alcohol	3	
4	percentage expenditure	389	
5	Hepatitis B	222	
6	Measles	542	
7	under-five deaths	394	
8	Polio	279	
9	Total expenditure	51	
10	Diphtheria	298	
11	HIV/AIDS	542	
12	GDP	300	
13	Population	203	
14	thinness 1-19 years	100	
15	thinness 5-9 years	99	
16	Income composition of resources	130	
17	Schooling	77	

	Percentage Before Winsorization	Outliers After Winsorization	\
0	0.578625	0	
1	2.927161	0	
2	10.721579	0	
3	0.102110	0	
4	13.240300	0	
5	7.556161	0	
6	18.447924	0	
7	13.410483	0	
8	9.496256	0	

```
9                            1.735875                                    0
10                          10.142954                                    0
11                          18.447924                                    0
12                          10.211028                                    0
13                           6.909462                                    0
14                           3.403676                                    0
15                           3.369639                                    0
16                           4.424779                                    0
17                           2.620830                                    0

    Percentage After Winsorization
0                                0.0
1                                0.0
2                                0.0
3                                0.0
4                                0.0
5                                0.0
6                                0.0
7                                0.0
8                                0.0
9                                0.0
10                               0.0
11                               0.0
12                               0.0
13                               0.0
14                               0.0
15                               0.0
16                               0.0
17                               0.0
```

Display the new DataFrame with winsorized data

```
display(winsorized_df.head())

   winsorized_Life expectancy  winsorized_Adult Mortality  \
0                        65.0                       263.0
1                        77.8                        74.0
2                        75.6                        19.0
3                        52.4                       335.0
4                        76.4                        13.0

   winsorized_infant deaths  winsorized_Alcohol  \
0                        52            0.010000
1                         0            4.600000
2                        21            5.288333
3                        52            5.288333
4                         0            5.288333

   winsorized_percentage expenditure  winsorized_Hepatitis B  \
0                          71.279624                    65.0
```

```
1                          364.975229                    99.0
2                            0.000000                    95.0
3                            0.000000                    64.0
4                            0.000000                    99.0

   winsorized_Measles  winsorized_under-five deaths  winsorized_Polio
\
0                 831                            61               51.0

1                   0                             0               99.0

2                  63                            24               95.0

3                 118                            61               51.0

4                   0                             0               86.0


   winsorized_Total expenditure  winsorized_Diphtheria
winsorized_HIV/AIDS  \
0                          8.16              73.627778
0.1
1                          6.00              99.000000
0.1
2                          7.08              95.000000
0.1
3                          7.08              73.627778
1.7
4                          7.08              99.000000
0.2

   winsorized_GDP  winsorized_Population  winsorized_thinness  1-19
years  \
0      584.259210           1.828185e+07
14.6
1     3954.227830           2.887300e+04
1.2
2     4132.762920           1.828185e+07
6.0
3     3695.793748           2.785935e+06
8.3
4    13235.977570           1.109741e+07
3.3

   winsorized_thinness 5-9 years  winsorized_Income composition of
resources  \
0                          15.0
0.479
1                           1.3
0.762
```

```
2                                5.8
0.743
3                                8.2
0.531
4                                3.3
0.784

   winsorized_Schooling
0                  10.1
1                  14.2
2                  14.4
3                  11.4
4                  13.9
```

Plot the histogram (Before Winsorization)

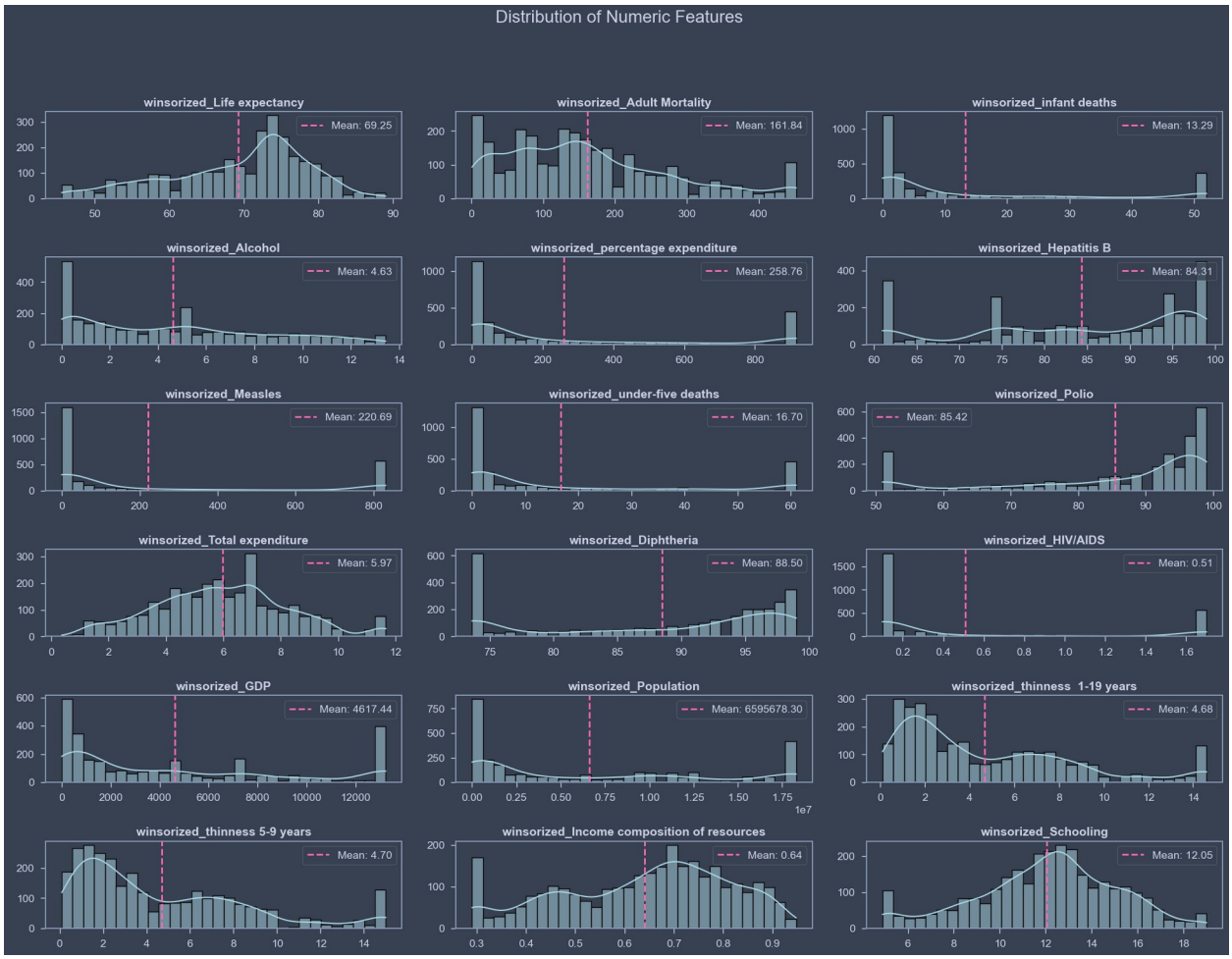
```
# Ignore future warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Set the color scheme
hist_color = 'lightblue'
line_color = 'hotpink'

# Create a figure
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(20, 15))
fig.suptitle("Distribution of Numeric Features", fontsize=20, y=1.02)

# Get numeric columns
numeric_cols =
life_expectancy_df.select_dtypes(exclude=['object']).columns.tolist()

for col, ax in zip(numeric_cols, axes.flatten()):
    # Plot histogram with KDE line
    sns.histplot(life_expectancy_df[col], bins=30, color=hist_color,
kde=True, ax=ax, edgecolor='black', linewidth=1)

    # Title for each subplot
    ax.set_title(col, fontsize=14, fontweight='bold')

    # Set axis labels
    ax.set_xlabel('')
    ax.set_ylabel('')

    # Add vertical line at mean if desired
    mean_value = life_expectancy_df[col].mean()
    ax.axvline(mean_value, color=line_color, linestyle='dashed',
linewidth=2, label=f'Mean: {mean_value:.2f}')
    ax.legend()

# Adjust layout for better spacing
```

```
plt.tight_layout(rect=[0, 0, 1, 0.97])

# Show the plot
plt.show()
```

Plot the histogram (After Winsorization)

```
# Ignore future warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Set the color scheme
hist_color = 'lightblue'
line_color = 'hotpink'

# Create a figure
fig, axes = plt.subplots(nrows = 6, ncols = 3, figsize=(20, 15))
fig.suptitle("Distribution of Numeric Features", fontsize=20, y=1.02)

# Get numeric columns
numeric_cols =
winsorized_df.select_dtypes(exclude=['object']).columns.tolist()
```

```
for col, ax in zip(numeric_cols, axes.flatten()):
    # Plot histogram with KDE line
    sns.histplot(winsorized_df[col], bins=30, color=hist_color,
kde=True, ax=ax, edgecolor='black', linewidth=1)

    # Title for each subplot
    ax.set_title(col, fontsize=14, fontweight='bold')

    # Set axis labels
    ax.set_xlabel('')
    ax.set_ylabel('')

    # Add vertical line at mean if desired
    mean_value = winsorized_df[col].mean()
    ax.axvline(mean_value, color=line_color, linestyle='dashed',
linewidth=2, label=f'Mean: {mean_value:.2f}')
    ax.legend()

# Adjust layout for better spacing
plt.tight_layout(rect=[0, 0, 1, 0.97])

# Show the plot
plt.show()
```

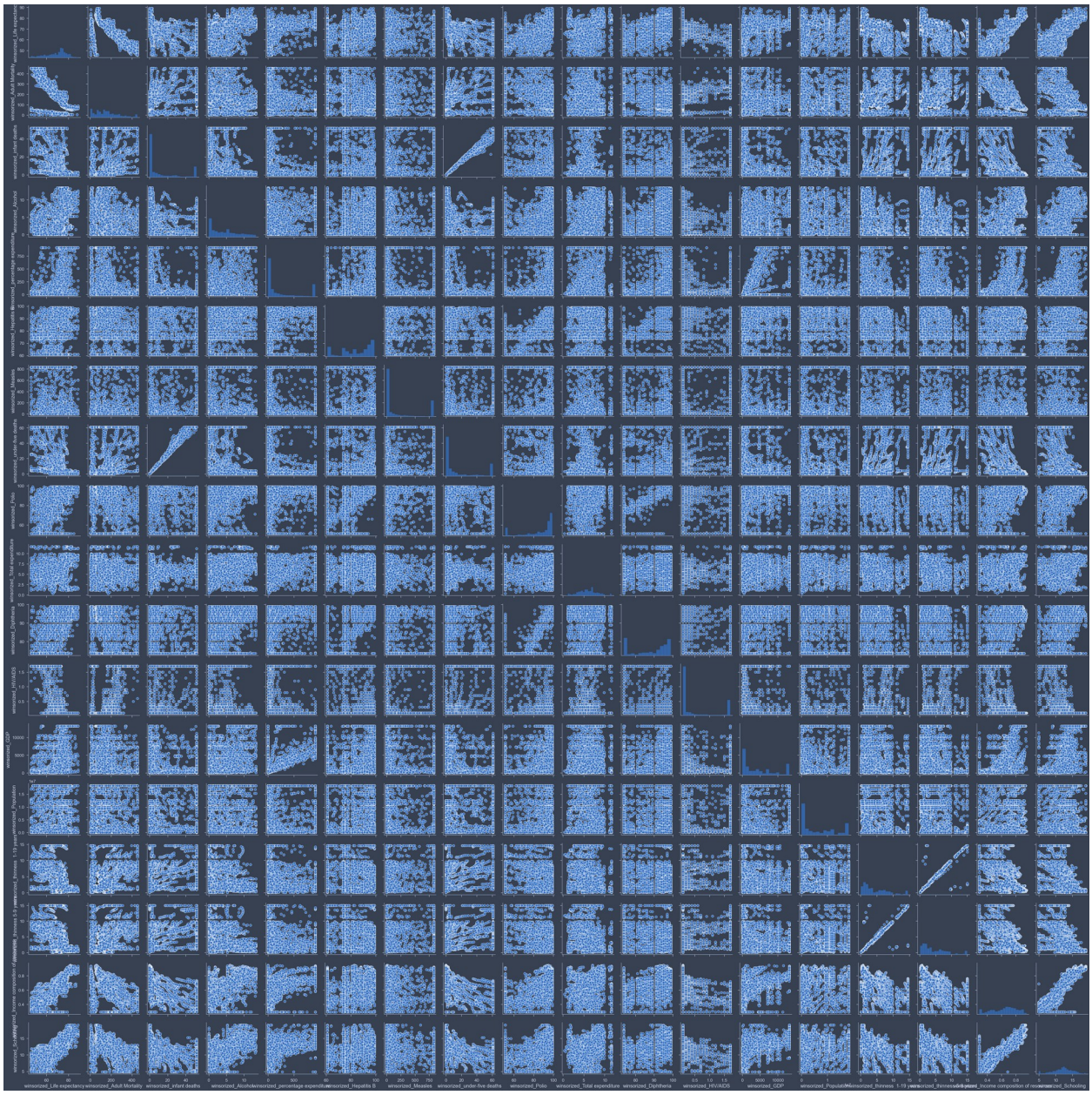


Plot Pairplot

```
plt.figure(figsize = (20, 20))
sns.pairplot(winsorized_df)
```

```
<seaborn.axisgrid.PairGrid at 0x2c374298fd0>
```

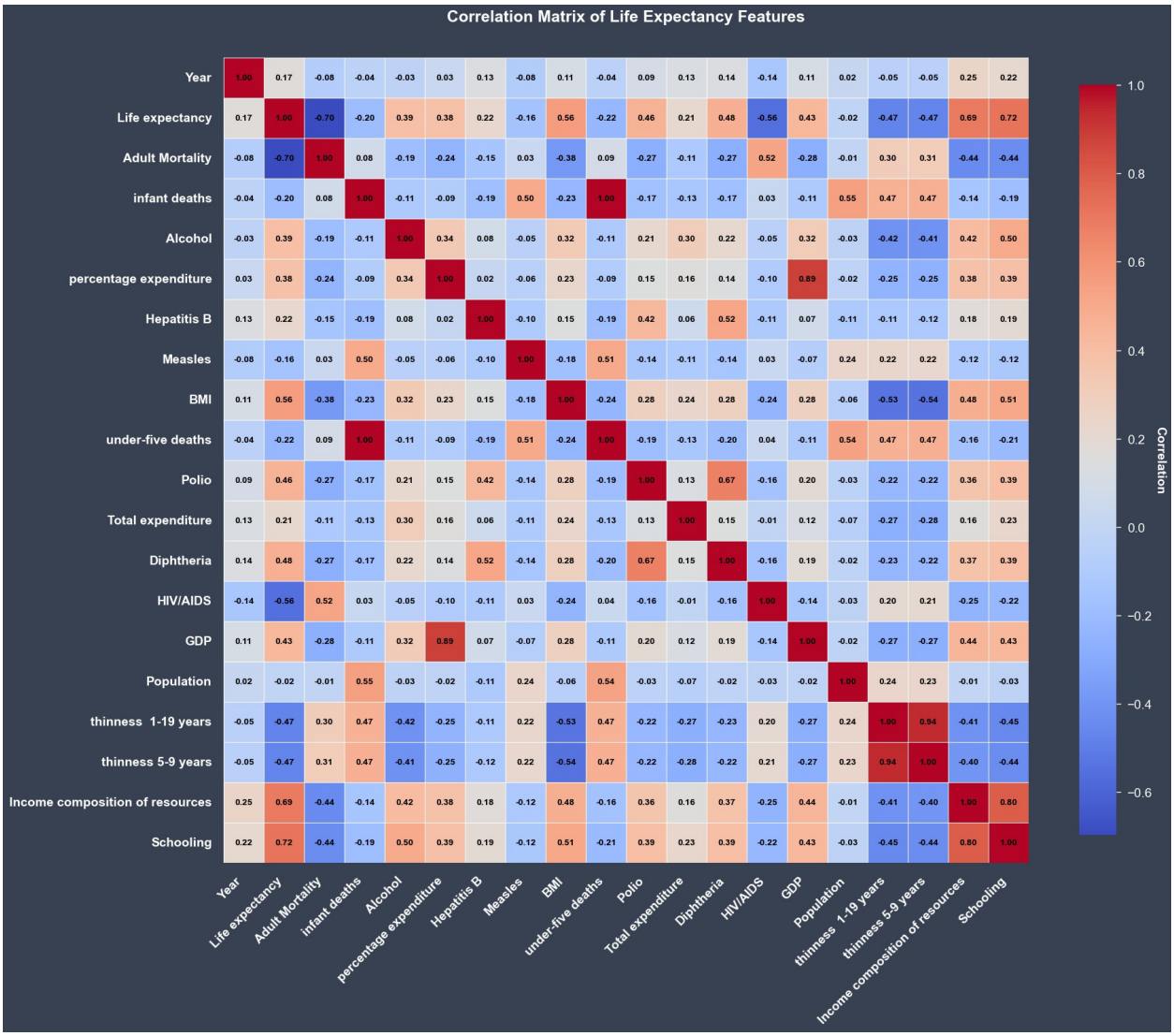
```
<Figure size 2000x2000 with 0 Axes>
```



Plot the correlation Matrix (Before Winsorization)

```
# Set the Jupyter theme
jtplot.style(theme='oceans16', context='notebook', ticks=True,
grid=False)


# Set figure size with adjusted aspect ratio
plt.figure(figsize=(20, 20))

# Select only numeric columns
numeric_columns = life_expectancy_df.select_dtypes(include=['float64',
'int64']).columns
```

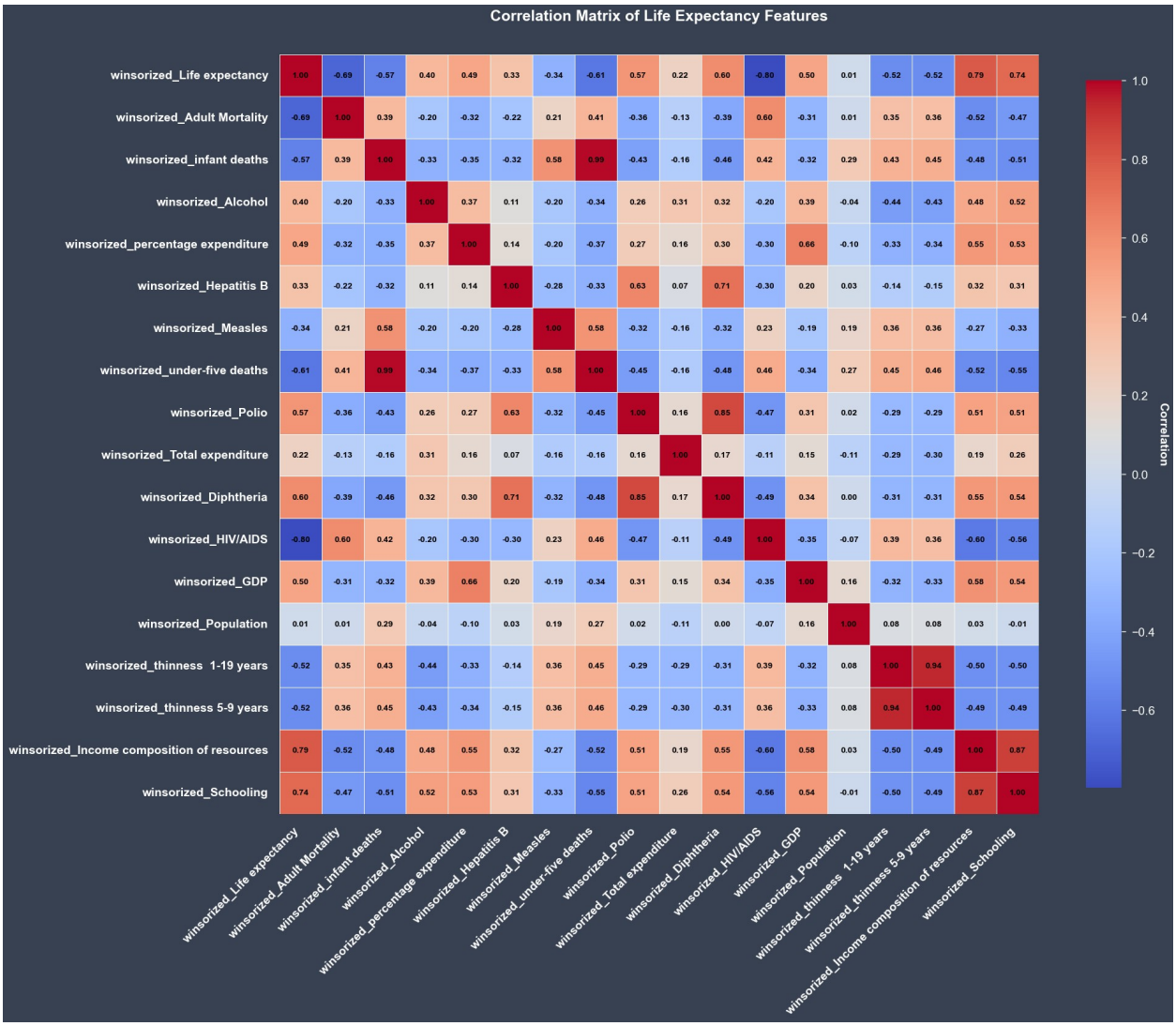
```
numeric_df = life_expectancy_df[numeric_columns]

# Calculate correlation matrix
corr_matrix = numeric_df.corr()

# Set font scale for the entire plot
sns.set(font_scale=1.2)

# Create heatmap without annotations
heatmap = sns.heatmap(corr_matrix, cmap="coolwarm", linewidths=.5,
cbar_kws={"shrink": 0.75}, square=True)

# Customize xticks and yticks
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=45,
horizontalalignment='right', fontsize=14, fontweight="bold",
color='white')
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0,
fontsize=14, fontweight="bold", color='white')

# Customize colorbar label
cbar = heatmap.collections[0].colorbar
cbar.set_label("Correlation", rotation=270, labelpad=15, fontsize=14,
fontweight="bold", color='white')

# Set color and fontweight for colorbar tick labels
cbar.ax.yaxis.set_tick_params(color='white', labelcolor='white',
labelsize=14, width=1, size=5, which='both')

# Loop to display annotations in all cells
for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix.columns)):
        text = f"{corr_matrix.iloc[i, j]:.2f}"
        plt.text(j + 0.5, i + 0.5, text, fontsize=10, ha='center',
va='center', color='black', fontweight='bold')

# Adjust the position of the title
plt.title("Correlation Matrix of Life Expectancy Features",
loc="center", pad=40, fontsize=18, fontweight="bold", color='white')

# Show the plot
plt.show()
```



Plot the correlation Matrix (After Winsorization)

```
# Set the Jupyter theme
jtplot.style(theme='oceans16', context='notebook', ticks=True,
grid=False)


# Set figure size with adjusted aspect ratio
plt.figure(figsize=(20, 20))

# Select only numeric columns
numeric_columns = winsorized_df.select_dtypes(include=['float64',
'int64']).columns
numeric_df = winsorized_df[numeric_columns]

# Calculate correlation matrix
corr_matrix = numeric_df.corr()
```

```
# Set font scale for the entire plot
sns.set(font_scale=1.2)

# Create heatmap without annotations
heatmap = sns.heatmap(corr_matrix, cmap="coolwarm", linewidths=.5,
cbar_kws={"shrink": 0.75}, square=True)

# Customize xticks and yticks
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=45,
horizontalalignment='right', fontsize=14, fontweight="bold",
color='white')
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0,
fontsize=14, fontweight="bold", color='white')

# Customize colorbar label
cbar = heatmap.collections[0].colorbar
cbar.set_label("Correlation", rotation=270, labelpad=15, fontsize=14,
fontweight="bold", color='white')

# Set color and fontweight for colorbar tick labels
cbar.ax.yaxis.set_tick_params(color='white', labelcolor='white',
labelsize=14, width=1, size=5, which='both')

# Loop to display annotations in all cells
for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix.columns)):
        text = f"{corr_matrix.iloc[i, j]:.2f}"
        plt.text(j + 0.5, i + 0.5, text, fontsize=10, ha='center',
va='center', color='black', fontweight='bold')

# Adjust the position of the title
plt.title("Correlation Matrix of Life Expectancy Features",
loc="center", pad=40, fontsize=18, fontweight="bold", color='white')

# Show the plot
plt.show()
```



Make a new datafram consists of Categorical Variables and Columns with high correlation with Life Expectancy

```
LE_df = pd.concat([life_expectancy_df[['Year', 'Status']],
                   winsorized_df[['winsorized_Schooling',
'winsorized_Income composition of resources', 'winsorized_GDP',
                                  'winsorized_HIV/AIDS',
'winsorized_Polio', 'winsorized_under-five deaths', 'winsorized_Adult
Mortality', 'winsorized_Life expectancy']]],
                  axis=1)

# Rename columns without 'winsorized_'
LE_df = LE_df.rename(columns={'winsorized_Schooling': 'Schooling',
                              'winsorized_Income composition of
resources': 'Income Composition of Resources',
                              'winsorized_GDP': 'GDP',
                              'winsorized_HIV/AIDS': 'HIV/AIDS',
```

```
                                'winsorized_Polio': 'Polio',
                                'winsorized_under-five deaths': 'under-
five deaths',
                                'winsorized_Adult Mortality': 'Adult
Mortality',
                                'winsorized_Life expectancy': 'Life
Expectancy'})

# Display the result DataFrame
display(LE_df)
```

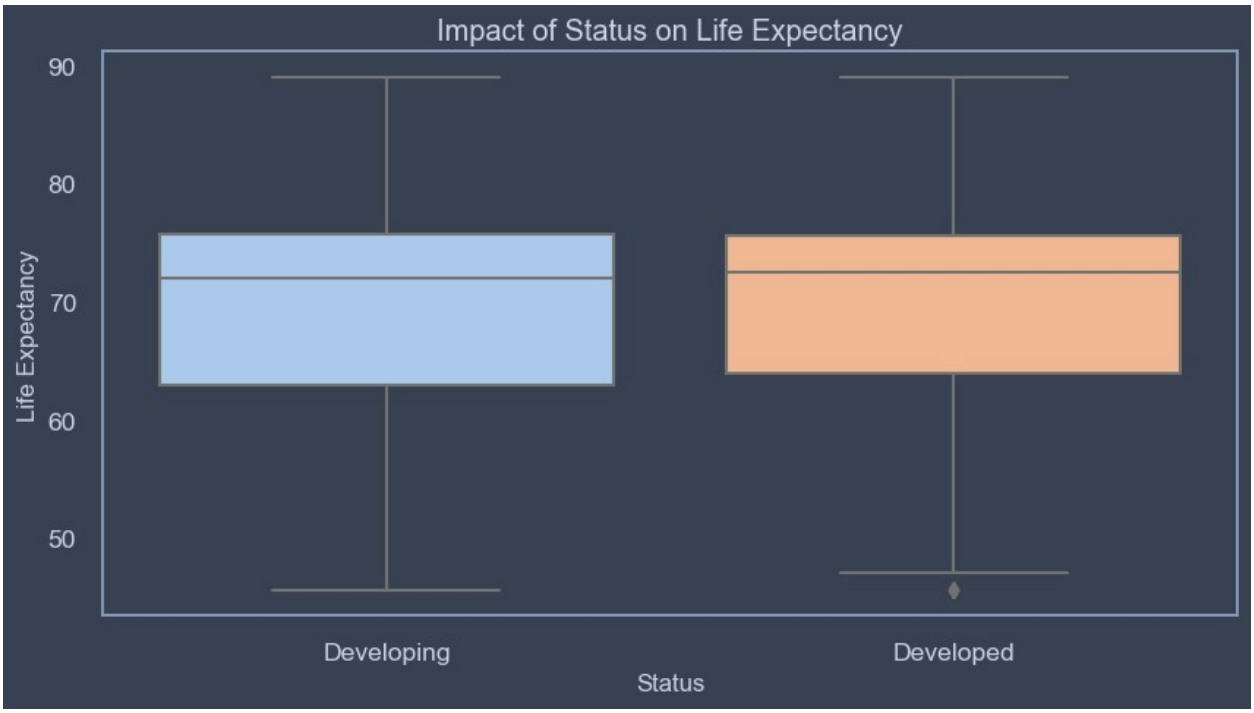
```
      Year      Status  Schooling  Income Composition of Resources  \
0     2015  Developing       10.1                             0.479
16    2015  Developing       12.8                             0.706
32    2015  Developing        7.1                             0.347
48    2015  Developing       13.2                             0.718
64    2015  Developing       17.2                             0.865
...    ...         ...        ...                               ...
2873  2000  Developing        7.6                             0.291
2889  2000  Developing        6.6                             0.318
2905  2000  Developing        4.9                             0.291
2921  2000  Developing       12.8                             0.646
2937  2000  Developing        9.8                             0.434

                GDP  HIV/AIDS      Polio  under-five deaths  Adult
Mortality  \
0       584.259210       0.1  51.000000                 61
263.0
16     4849.997495       0.2  94.000000                  0
175.0
32      348.381417       1.7  51.000000                 21
397.0
48     6468.471648       0.3  87.000000                  7
152.0
64      187.789910       0.1  99.000000                  0
72.0
...            ...       ...        ...                ...
...
2873    379.119326       1.7  51.000000                 61
45.0
2889    216.172747       1.7  51.000000                 57
426.0
2905   4708.515191       1.7  76.277778                 50
38.0
2921   2213.914880       0.1  97.000000                  5
112.0
2937    547.358878       1.7  78.000000                 39
452.0

      Life Expectancy
```

```
0                 65.0
16                71.0
32                52.5
48                73.9
64                81.0
...                ...
2873              47.1
2889              48.3
2905              48.9
2921              72.9
2937              46.0

[2938 rows x 10 columns]
```

The Winsorized Infant Death rate is also one of the columns that has a high correlation with Life Expectancy. However, as it has a strong correlation with under-five deaths, we choose one of them for our modeling.

Descriptive statistics of categorical variables

```
# Set the Jupyter theme
jtplot.style(theme='oceans16', context='notebook', ticks=True,
grid=False)

# Set figure size
plt.figure(figsize=(10, 5))

# Create a box plot
sns.boxplot(x='Status', y='Life Expectancy', data=LE_df,
palette='pastel')
plt.xlabel("Status", fontsize=12)
plt.ylabel("Life Expectancy", fontsize=12)
plt.title("Impact of Status on Life Expectancy")
plt.show()
```




```
# Finding the significance of difference of Average_Life_Expectancy
between Developed and Developing countries using
# t-test
import scipy.stats as stats
stats.ttest_ind(LE_df.loc[LE_df['Status']=='Developed','Life
Expectancy'],LE_df.loc[LE_df['Status']=='Developing','Life
Expectancy'])
```

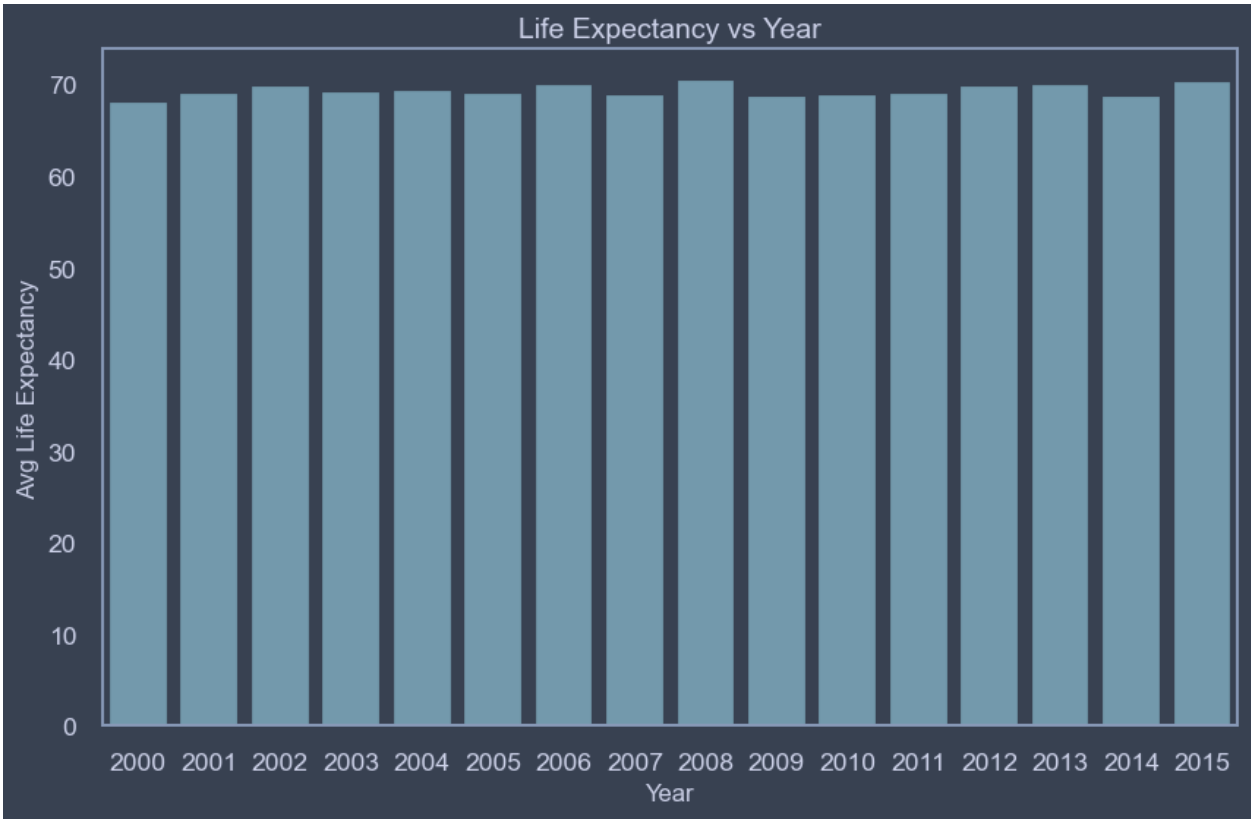
```
TtestResult(statistic=0.35931342286299406, pvalue=0.719386472985363,
df=2936.0)
```

p value is > 0.05

Hence, the difference of Average_Life_Expectancy between Developed and Developing countries is not significant. We can not consider 'Status' as a feature as it is related to Life Expectancy.

```
plt.figure(figsize=(10, 6))

# Create the bar plot
sns.barplot(x=LE_df.groupby('Year')['Year'].count().index,
            y=LE_df.groupby('Year')['Life Expectancy'].mean(),
            color='skyblue', alpha=0.65)

plt.xlabel("Year", fontsize=12)
plt.ylabel("Avg Life Expectancy", fontsize=12)
plt.title("Life Expectancy vs Year", fontsize=14)  # Adjusting title
font size
```

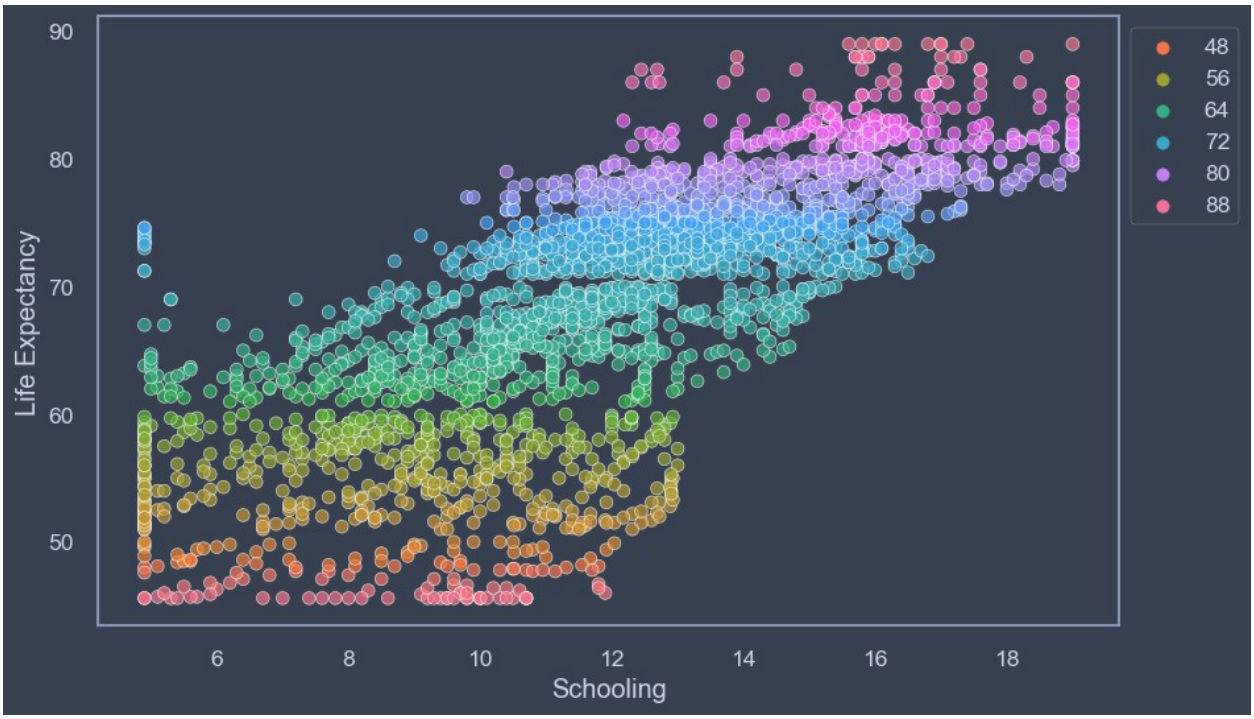
```
# Display the plot
plt.show()
```



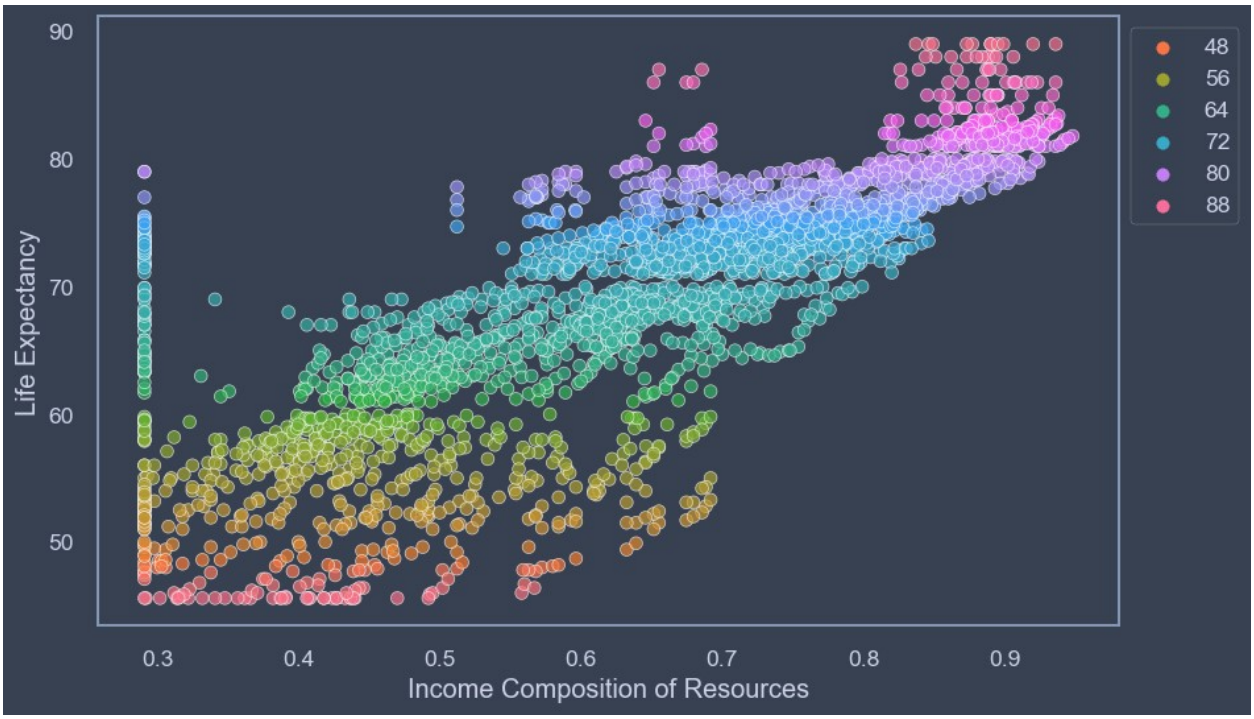

```
# Set the Jupyter theme
jtplot.style(theme='oceans16', context='notebook', ticks=True,
grid=False)
```

Showing variables with a high correlation rate with Life Expectancy

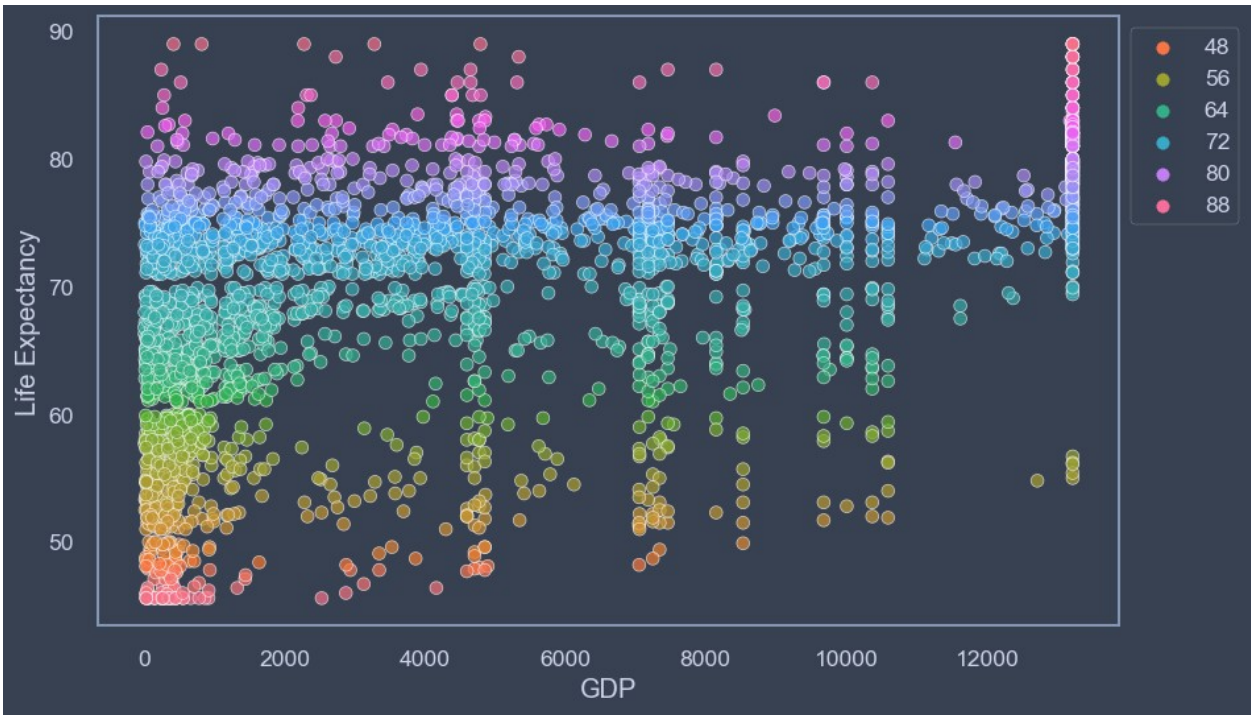
```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='Schooling', y='Life
Expectancy', hue='Life Expectancy', palette='husl', markers=['o'],
alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



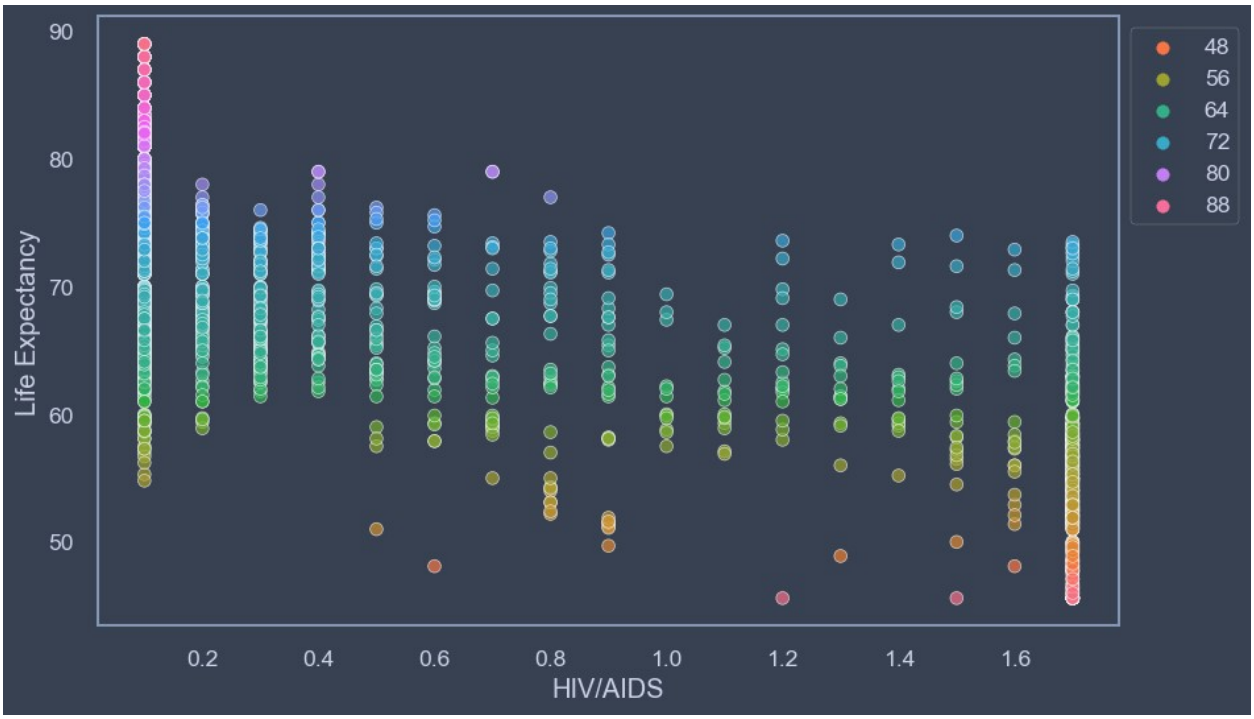

```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='Income Composition of
Resources', y='Life Expectancy', hue='Life Expectancy',
palette='husl', markers=['o'], alpha=0.7, edgecolor='white',
linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



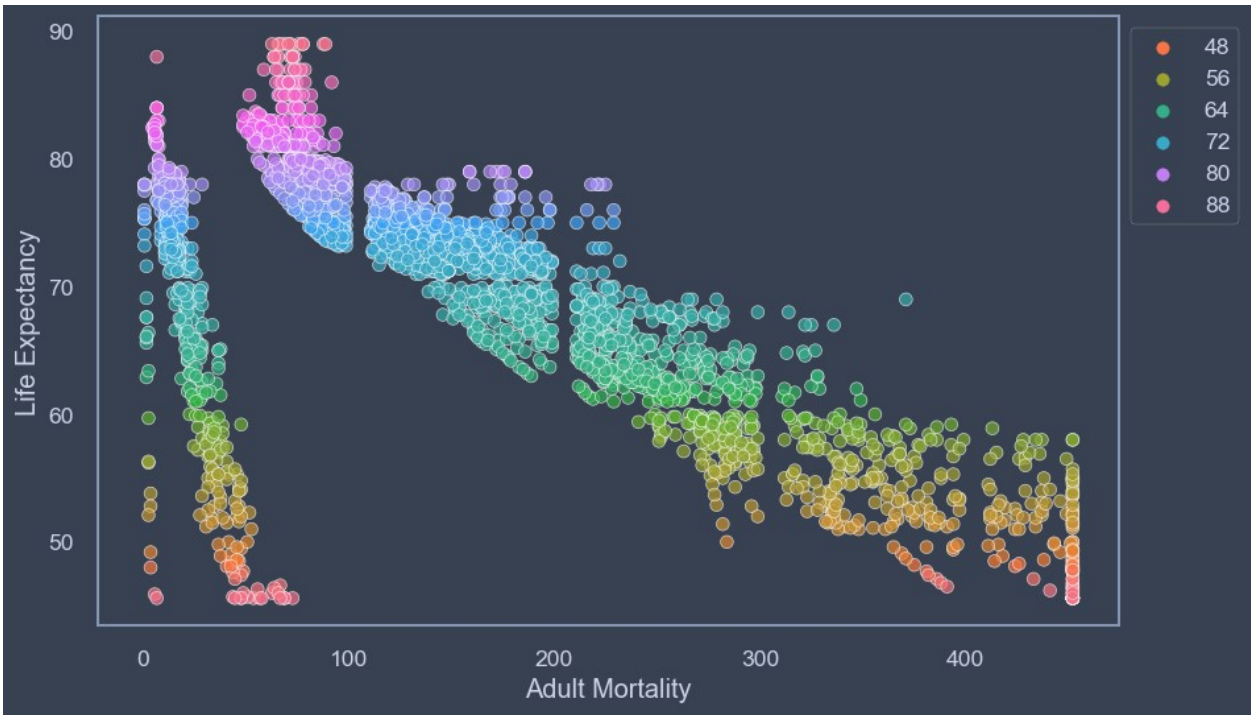

```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='GDP', y='Life
Expectancy', hue='Life Expectancy', palette='husl', markers=['o'],
alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



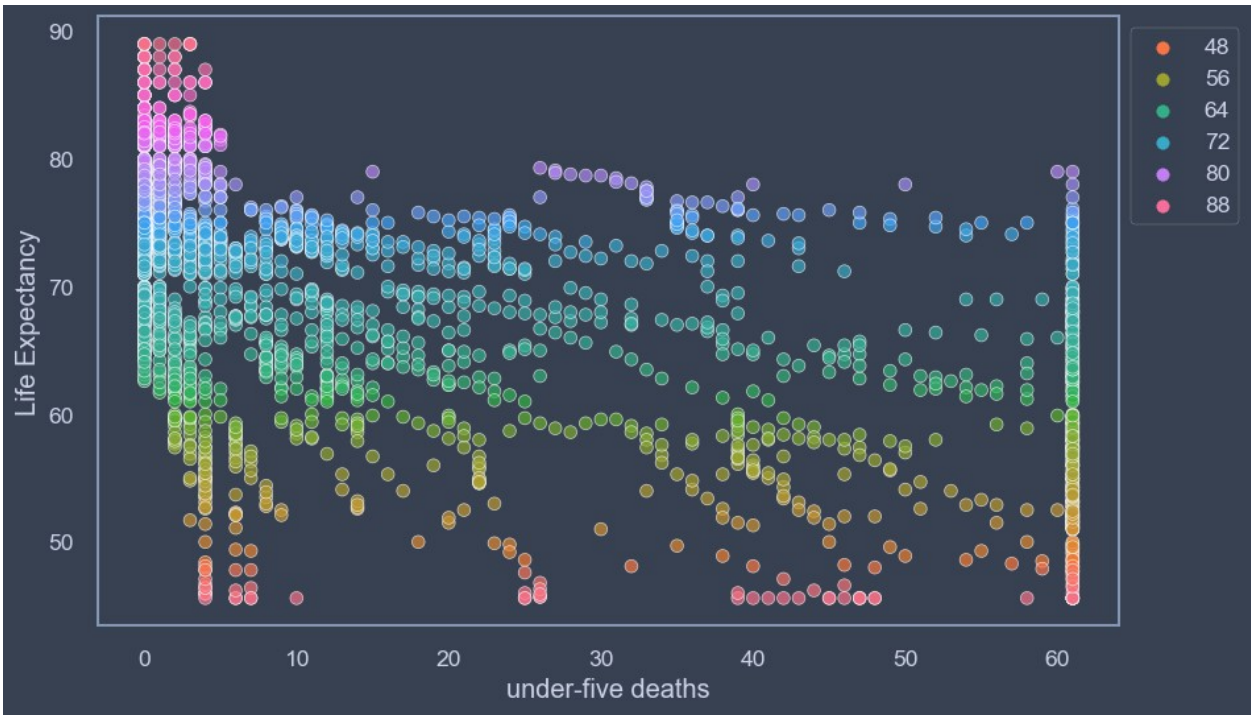

```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='HIV/AIDS', y='Life
Expectancy', hue='Life Expectancy', palette='husl', markers=['o'],
alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



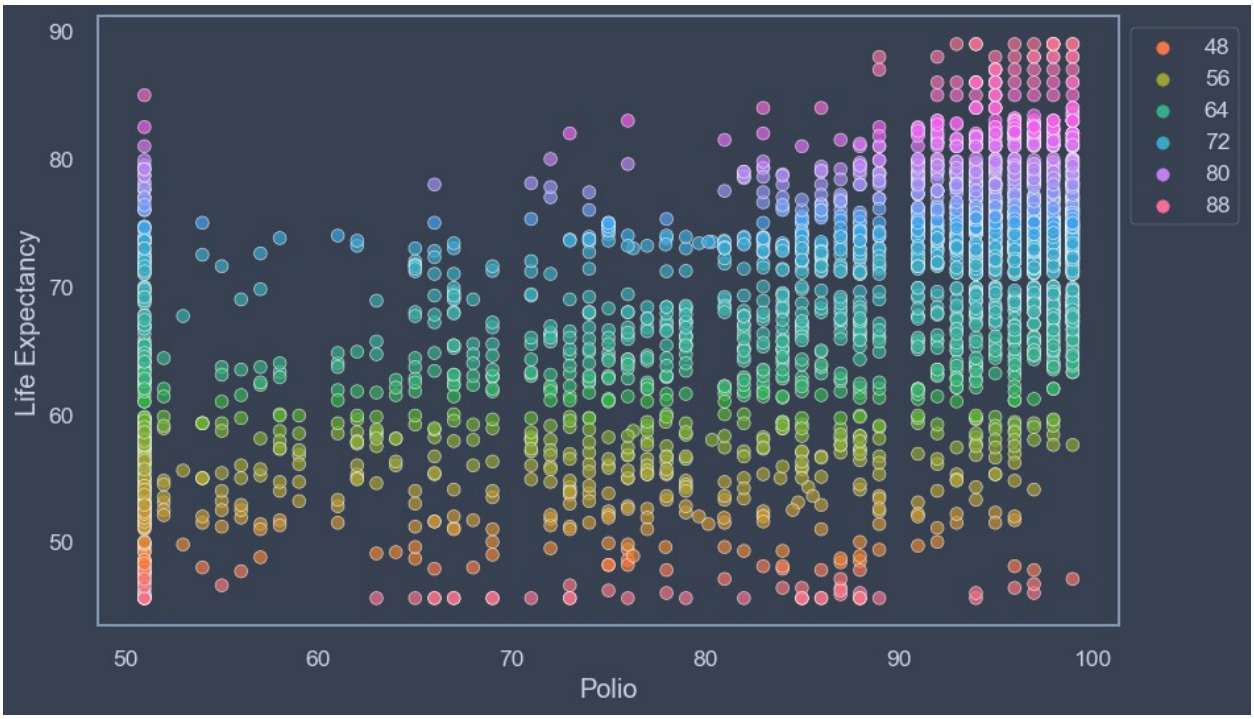

```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='Adult Mortality',
y='Life Expectancy', hue='Life Expectancy', palette='husl',
markers=['o'], alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```




```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='under-five deaths',
y='Life Expectancy', hue='Life Expectancy', palette='husl',
markers=['o'], alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```




```
# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter_plot = sns.scatterplot(data=LE_df, x='Polio', y='Life
Expectancy', hue='Life Expectancy', palette='husl', markers=['o'],
alpha=0.7, edgecolor='white', linewidth=0.5)

# Move the legend outside to the right of the plot
scatter_plot.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



Make Model

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,r2_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR

# Splitting testing into Indipendent and Dependent Variable

X = LE_df[['Schooling', 'Income Composition of Resources', 'GDP',
'HIV/AIDS', 'Polio', 'under-five deaths', 'Adult Mortality']]
y = LE_df['Life Expectancy']
```

Min-Max Scaling for Feature Normalization

```
from sklearn.preprocessing import MinMaxScaler

# Min-Max scaling
min_max_scaler = MinMaxScaler()
X_minmax_scaled = min_max_scaler.fit_transform(X)

# Create DataFrames for the scaled features
X_minmax_scaled_df = pd.DataFrame(X_minmax_scaled, columns=X.columns)
```

Now, X_minmax_scaled_df contains the Min-Max scaled features

```
X_minmax_scaled_df.head()
```

```
   Schooling  Income Composition of Resources       GDP  HIV/AIDS
Polio  \
0   0.368794                          0.286149  0.044020    0.0000
0.000000
1   0.560284                          0.631659  0.366345    0.0625
0.895833
2   0.156028                          0.085236  0.026197    1.0000
0.000000
3   0.588652                          0.649924  0.488639    0.1250
0.750000
4   0.872340                          0.873668  0.014063    0.0000
1.000000

   under-five deaths  Adult Mortality
0           1.000000         0.580931
1           0.000000         0.385809
2           0.344262         0.878049
3           0.114754         0.334812
4           0.000000         0.157428

y.head()

0     65.0
16    71.0
32    52.5
48    73.9
64    81.0
Name: Life Expectancy, dtype: float64
```

Splitting data into Train and Test sets

```
X_train, X_test, y_train, y_test = train_test_split(X_minmax_scaled,
y, test_size=0.30, random_state=101)
```

Model Selection and Evaluation Framework

```
# Create an empty DataFrame for modeling
Modeling = pd.DataFrame(columns=['Model', 'Training Score', 'Test R2
Score'])

# Define the model selection function
def select_model(model, Train, Target, x_test, y_test):
    global Modeling  # Access the global DataFrame

    # Fit the model on the training data
    model.fit(Train, Target)

    # Calculate the score of the model on the training data
    train_score = model.score(Train, Target)
    print(f"Score of the {type(model).__name__} model on the training
```

```
data is: {train_score}")

    # Make predictions on the test data
    predictions = np.round(model.predict(x_test), decimals=1)

    # See R2 score on the test data
    test_r2_score = r2_score(y_test, predictions)
    print(f"R2 score of the {type(model).__name__} model on the test
data is: {test_r2_score}")

    # Create a DataFrame for the current model's scores
    model_scores = pd.DataFrame({'Model': [type(model).__name__],
'Training Score': [train_score], 'Test R2 Score': [test_r2_score]})

    # Concatenate the model_scores DataFrame with the Modeling
DataFrame
    Modeling = pd.concat([Modeling, model_scores], ignore_index=True)
```

LinearRegression Model

```
# Instantiate the LinearRegression model
model = LinearRegression()

# Call the function with LinearRegression model
select_model(model, X_train, y_train, X_test, y_test)
```

```
Score of the LinearRegression model on the training data is:
0.8477897193708867
R2 score of the LinearRegression model on the test data is:
0.8294017253940305
```

SupportVectorRegression Model

```
# Instantiate the SupportVectorRegression model with specific
parameters
svr_model = SVR(C=9.0, epsilon=0.9, kernel='rbf')

# Call the function with SupportVectorRegression model
select_model(svr_model, X_train, y_train, X_test, y_test)
```

```
Score of the SVR model on the training data is: 0.9222027487858514
R2 score of the SVR model on the test data is: 0.9100067303188345
```

RandomForestRegressor Model

```
# Instantiate the RandomForestRegressor model with specific parameters
rf_model = RandomForestRegressor(n_estimators=100, max_depth=7,
min_samples_split=5)

# Call the function with RandomForestRegressor model
select_model(rf_model, X_train, y_train, X_test, y_test)
```

```
Score of the RandomForestRegressor model on the training data is:
0.9604180452959549
R2 score of the RandomForestRegressor model on the test data is:
0.9345294241821204
```

GradientBoostingRegressor Model

```
# Instantiate the GradientBoostingRegressor model with specific
parameters
gb_model = GradientBoostingRegressor(n_estimators=100, max_depth=6,
min_samples_split=5)

# Call the function with GradientBoostingRegressor model
select_model(gb_model, X_train, y_train, X_test, y_test)
```

```
Score of the GradientBoostingRegressor model on the training data is:
0.990258179149136
R2 score of the GradientBoostingRegressor model on the test data is:
0.9538796959708531
```

rename model name

```
names = ['LinearRegression', 'SuportVectorRegression',
'RandomForestRegressor', 'GradientBoostingRegressor']

# loop over the list to rename the model column
Modeling['Model'] = names

# drop the original "Model" column
Modeling.drop(columns='Model', inplace=True)
```

Create a bar plot for both training and test scores

```
from jupyterthemes import jtplot
jtplot.style(theme = 'oceans16', context = 'notebook', ticks = True,
grid = False)
# setting the style of the notebook to be monokai theme
# this line of code is important to ensure that we are able to see the
x and y axes clearly
# If you don't run this code line, you will notice that the xlabel and
ylabel on any plot is black on black and it will be hard to see them.

fig, ax = plt.subplots(figsize=(12, 8))

# Bar width for better separation
bar_width = 0.35

# Plot training scores
ax.bar(Modeling.index - bar_width/2, Modeling['Training Score'],
bar_width, color='skyblue', label='Training Score')
```

```
# Plot test R2 scores
ax.bar(Modeling.index + bar_width/2, Modeling['Test R2 Score'],
bar_width, color='lightpink', label='Test R2 Score')

ax.set_xlabel("Model", fontsize=12)
ax.set_ylabel("Score", fontsize=12)
ax.set_title("Training and Test Scores for Different Models",
fontsize=14)
ax.set_xticks(Modeling.index)
ax.set_xticklabels([name for name in names], rotation=45, ha='right')
# Use provided names as labels

ax.legend()  # Display legend to differentiate training and test
scores

# Show the plot
plt.show()
```