Advanced Data Science-T1 Mini Project Report

Title:Analyzing Global Trends in Life Expectancy

Members -
1. UCE2021463 : Eesha Satvalekar
2. UCE2021470 : Mrunal Vibhute
3. UCE2021472 : Aditi Wagh

Introduction

Understanding the intricate relationship between a nation's GDP and life expectancy is pivotal for policymakers and researchers. This report aims to delve deep into the data, utilizing interactive visualizations to provide nuanced insights into this correlation across various countries. The primary objective is to uncover patterns and outliers, thereby informing policies and interventions geared towards improving healthcare and socio-economic conditions.

Processing the dataset

Importing all the necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats.mstats import winsorize
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
import os
%matplotlib inline
```

The code processes a dataset comprising data from various countries worldwide, aggregated by the World Health Organization (WHO). This dataset encompasses multiple indicators over a span of 15 years for 243 countries. These indicators represent a wide array of factors affecting life expectancy, including health-related, financial, and other socio-economic factors. In essence, the data consists of a time series of indicators for each country.The below code uploads, reads, and merges these separate datasets into a single DataFrame

```
import pandas as pd
from google.colab import files
import matplotlib.pyplot as plt
uploaded=files.upload()
import io
```

```
#importing the different datasets consisting of factors affecting life
expectancy for countries over several years
df1 = pd.read_csv(io.BytesIO(uploaded['dataset1.csv']))
df2 = pd.read_csv(io.BytesIO(uploaded['dataset2.csv']))
df3 = pd.read_csv(io.BytesIO(uploaded['dataset3.csv']))
#merging the dataframes
df4=pd.merge(df1,df2)
df=pd.merge(df4,df3)
```

To ensure that column names in a DataFrame follow a standardized format and are more user-friendly and error-resistant for subsequent data analysis and manipulation we took the original column names, performed the following operations on each column name, and assigned the modified names back to the DataFrame:

1. Strips leading and trailing whitespace from the column names.
2. Replaces double spaces with single spaces.
3. Replaces spaces with underscores.
4. Converts the column names to lowercase.

```
#Storing the original columns in a list
orig_cols = list(df.columns)
new_cols = []
#Iterating over each column name and replacing double space with single
space,and single space with an underscore symbol
for col in orig_cols:
    new_cols.append(col.strip().replace('  ', ' ').replace(' ',
'_').lower())
#Assigning back that columns in the dataframe
df.columns = new_cols
```

As stated in the description of each columns, it would be useful to change the name of the variable thinness_1-19_years to thinness_10-19_years as it is a more accurate depiction of what the variable means.

```
#Renaming the incorrect column name
df.rename(columns={'thinness_1-19_years':'thinness_10-19_years'},
inplace=True)
```

After performing these operations the dataframe looks like this:

We are using a box plot to examine several variables simultaneously, which can be useful for a quick overview of their distributions and to identify potential issues with outliers or extreme values.

```
#Plotting the box plots for some features
plt.figure(figsize=(15,10))
for i, col in enumerate(['adult_mortality', 'infant_deaths', 'bmi',
'under-five_deaths', 'gdp', 'population'], start=1):
    plt.subplot(2, 3, i)
    df.boxplot(col)
```

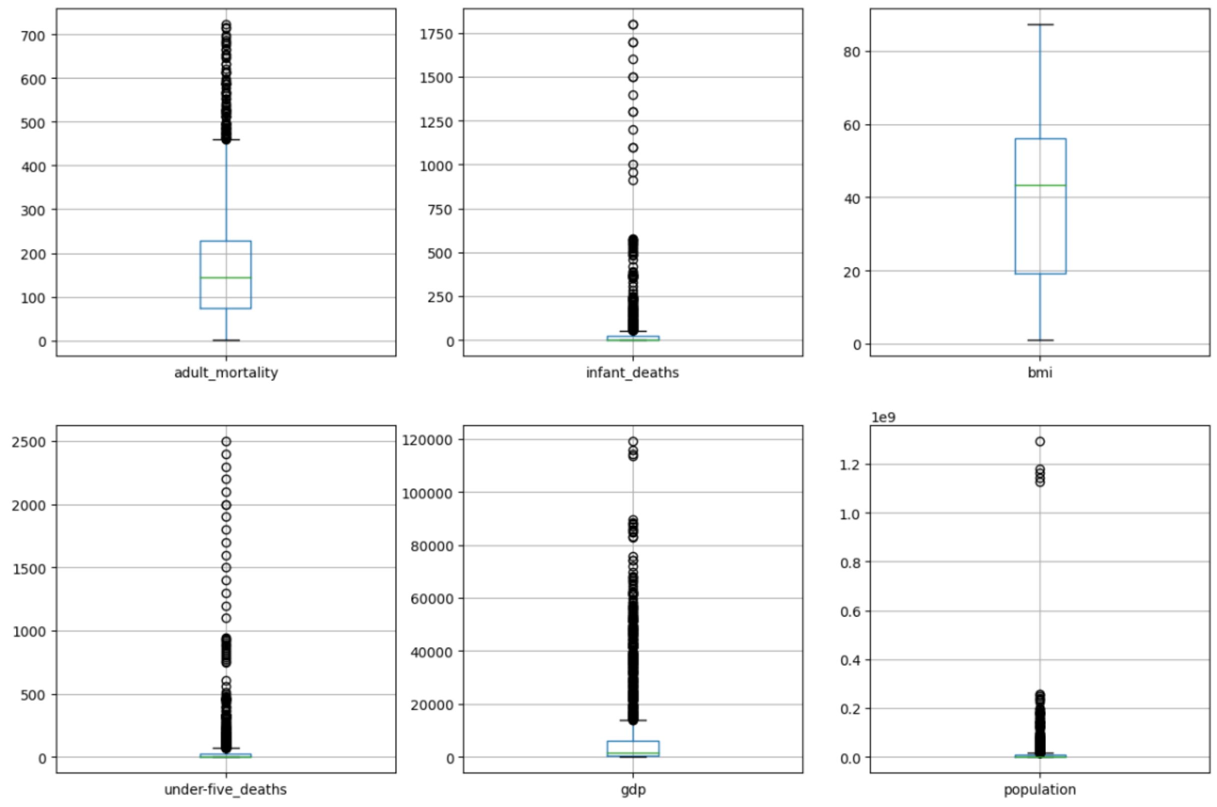


Figure:1 Boxplot for some features

From the above graphs we conclude that there are some values that may not make sense or could be erroneous.

1. Adult Mortality of 1: An Adult Mortality rate of 1 is highly unlikely and could indicate a measurement error. Typically, Adult Mortality rates are represented as a percentage, so values between 0 and 100 are more reasonable

2. Infant Deaths as low as 0 per 1000: While it's possible for some regions or time periods to report very low Infant Mortality rates, values as low as 0 per 1000 births are

uncommon. It's reasonable to treat values close to 0 as potentially erroneous and consider them as null or investigate data sources.

3. BMI of 1 and 87.3: Extremely low or high BMI values may be unrealistic for an entire population. A BMI of 1 would imply severe underweight, and 87.3 is beyond the extreme of morbid obesity. Investigating and potentially excluding or imputing these extreme values might be necessary.

4. Under Five Deaths at zero: Values of zero for Under-Five Deaths may not be plausible since child mortality, unfortunately, exists in most populations. Consider treating such zero values as potential data quality issues or investigating the data source for more context.
5. GDP per capita as low as 1.68 USD: While low GDP per capita values are possible, 1.68 USD is exceptionally low and might represent data outliers or errors. Investigating these values and potentially identifying if they are exceptional cases or data errors is advisable.

6. Population of 34 for an entire country: A population of 34 for an entire country is extremely low and would be highly unusual. It could be a data entry error, incomplete data, or it might represent a very small, specialized population group. Investigating this further and cross-referencing with reliable population data sources is necessary to determine the accuracy of this data point.

To handle extreme or implausible values, ensuring that the dataset used for analysis is more accurate and reliable we performed the following steps:

1. Calculate the 5th percentile of the "adult_mortality" column and replaces values below this threshold with NaN, effectively removing extremely low values.
2. Replacing all instances of 0 in the "infant_deaths" column with NaN, as having zero infant deaths is considered unlikely.
3. Checking the "bmi" column and replacing values below 10 or above 50 with NaN, as these values are considered unrealistic for Body Mass Index.
4. Replacing all instances of 0 in the "under-five_deaths" column with NaN, as having zero under-five deaths is improbable.

```
mort_5_percentile = np.percentile(df.adult_mortality.dropna(), 5)
df.adult_mortality = df.apply(lambda x: np.nan if x.adult_mortality <
mort_5_percentile else x.adult_mortality, axis=1)
df.infant_deaths = df.infant_deaths.replace(0, np.nan)
df.bmi = df.apply(lambda x: np.nan if (x.bmi < 10 or x.bmi > 50) else
x.bmi, axis=1)
```

```
df['under-five_deaths'] = df['under-five_deaths'].replace(0, np.nan)
```

The nulls_breakdown function is designed to analyze missing values in a DataFrame. It iterates through each column, counting the number of null values and calculating the percentage of null values in each column. The function then reports this information, including the column name, number of null values, and the percentage of null values. Additionally, it provides a summary at the end, stating the total number of columns with null values and the percentage of columns containing null values in the DataFrame.

```
def nulls_breakdown(df=df):
    df_cols = list(df.columns)
    cols_total_count = len(list(df.columns))
    cols_count = 0
    for loc, col in enumerate(df_cols):
        null_count = df[col].isnull().sum()
        total_count = df[col].isnull().count()
        percent_null = round(null_count/total_count*100, 2)
        if null_count > 0:
            cols_count += 1
            print('[iloc = {}] {} has {} null values: {}%
null'.format(loc, col, null_count, percent_null))
    cols_percent_null = round(cols_count/cols_total_count*100, 2)
    print('Out of {} total columns, {} contain null values; {}% columns
contain null values.'.format(cols_total_count, cols_count,
cols_percent_null))
```

After calling the function the output is:

```
[iloc = 4] total_expenditure has 226 null values: 7.69% null
[iloc = 5] gdp has 448 null values: 15.25% null
[iloc = 6] population has 652 null values: 22.19% null
[iloc = 7] income_composition_of_resources has 167 null values: 5.68% null
[iloc = 8] schooling has 163 null values: 5.55% null
[iloc = 9] life_expectancy has 10 null values: 0.34% null
[iloc = 10] adult_mortality has 155 null values: 5.28% null
[iloc = 11] infant_deaths has 848 null values: 28.86% null
[iloc = 12] under-five_deaths has 785 null values: 26.72% null
[iloc = 13] alcohol has 194 null values: 6.6% null
[iloc = 14] hepatitis_b has 553 null values: 18.82% null
[iloc = 16] bmi has 1456 null values: 49.56% null
[iloc = 17] polio has 19 null values: 0.65% null
[iloc = 18] diphtheria has 19 null values: 0.65% null
[iloc = 20] thinness_10-19_years has 34 null values: 1.16% null
[iloc = 21] thinness_5-9_years has 34 null values: 1.16% null
Out of 22 total columns, 16 contain null values; 72.73% columns contain null values.
```

Nearly half of the BMI variable's values are null, it is likely best to remove this variable altogether.

```
df.drop('bmi', axis=1, inplace=True)
```

The below code identifies columns in the DataFrame that have missing values within a specific range between 0% and 5%.

```
cols=[var for var in df.columns if df[var].isnull().mean()<0.05 and
df[var].isnull().mean()>0]
cols
```

We calculate the percentage of rows that remain after removing rows with missing values in the selected columns. This percentage is relative to the total number of rows in the original DataFrame.

```
(len(df[cols].dropna())/len(df))*100
```

Rows with missing values in the specified columns are dropped from the DataFrame to retain data with minimal missing data in those columns.

```
df.dropna(subset=cols,inplace=True)
```

After cleansing the data, the distribution of a specific column is plotted, allowing for a visual comparison of the data distribution before and after the removal of rows with missing values. This step helps to assess the impact of missing value handling on the distribution of the selected column and understand how the data has been affected.

```
fig = plt.figure()
```

```
ax = fig.add_subplot(111)
# original data
original_df['life_expectancy'].plot.density(color='red')
# data after cca
df['life_expectancy'].plot.density(color='green')
```

```
<Axes: ylabel='Density'>
```

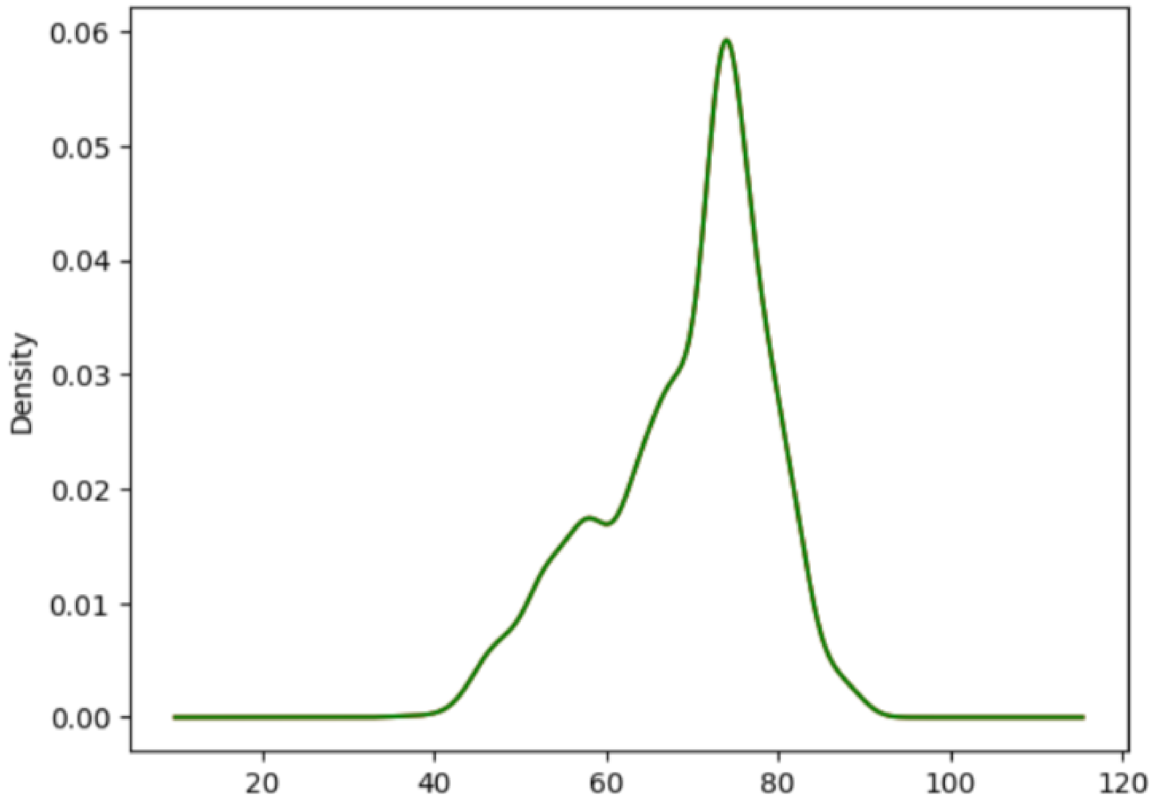


Figure:2 Density plot for life expectancy

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
original_df['adult_mortality'].plot.density(color='red')

# data after cca
df['adult_mortality'].plot.density(color='green')
```

```
<Axes: ylabel='Density'>
```

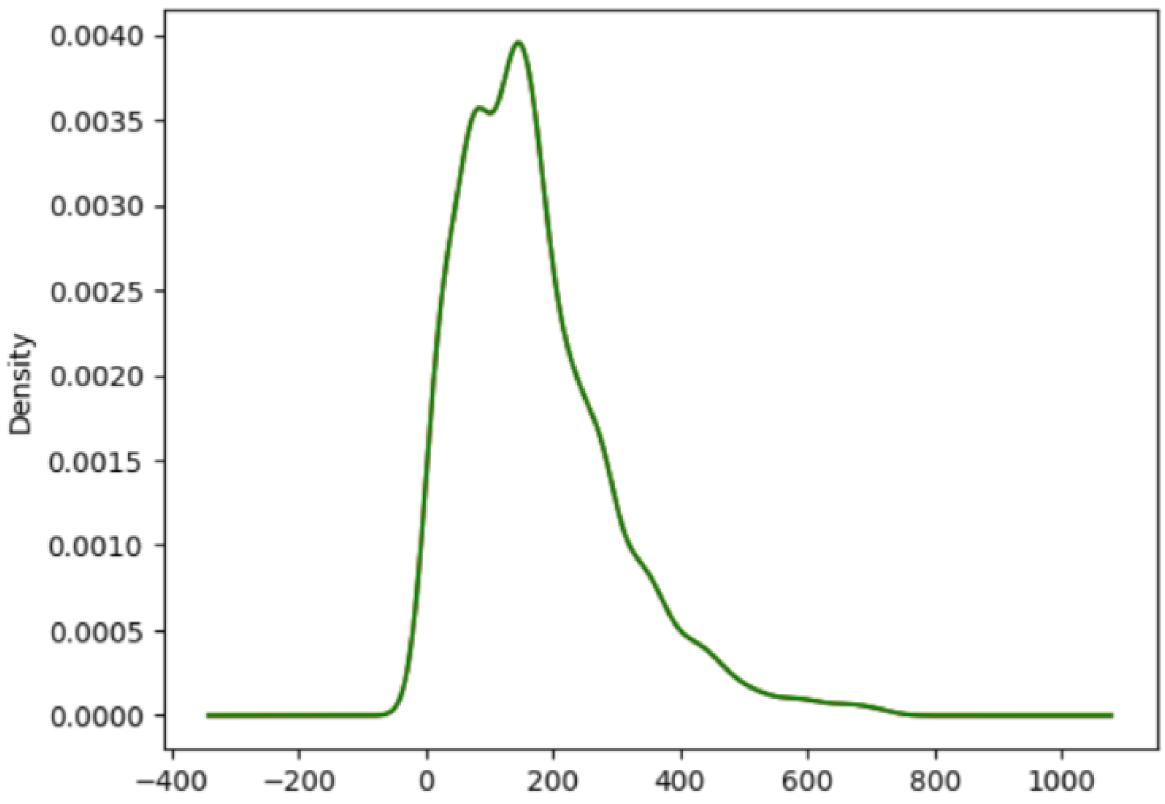


Figure:3 Density plot for adult mortality

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
original_df['polio'].plot.density(color='red')

# data after cca
df['polio'].plot.density(color='green')
```

```
<Axes: ylabel='Density'>
```

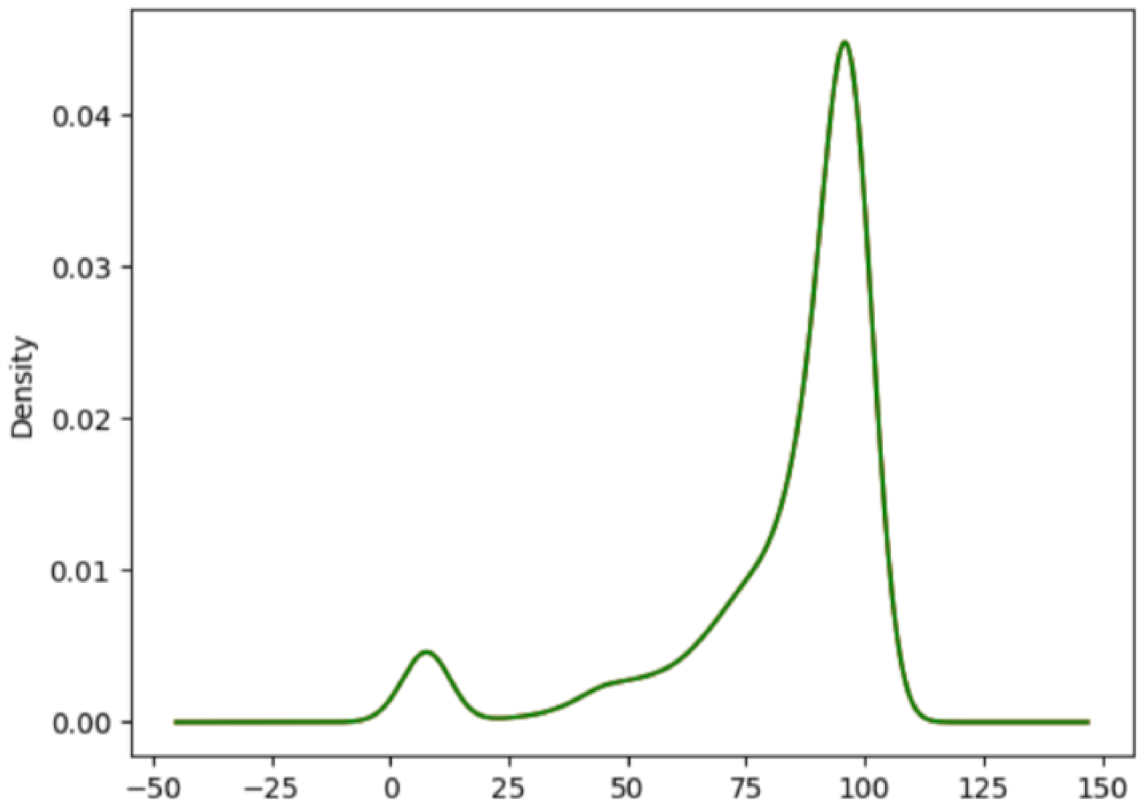


Figure:4 Density plot for polio

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
original_df['diphtheria'].plot.density(color='red')

# data after cca
df['diphtheria'].plot.density(color='green'
<Axes: ylabel='Density'>
```

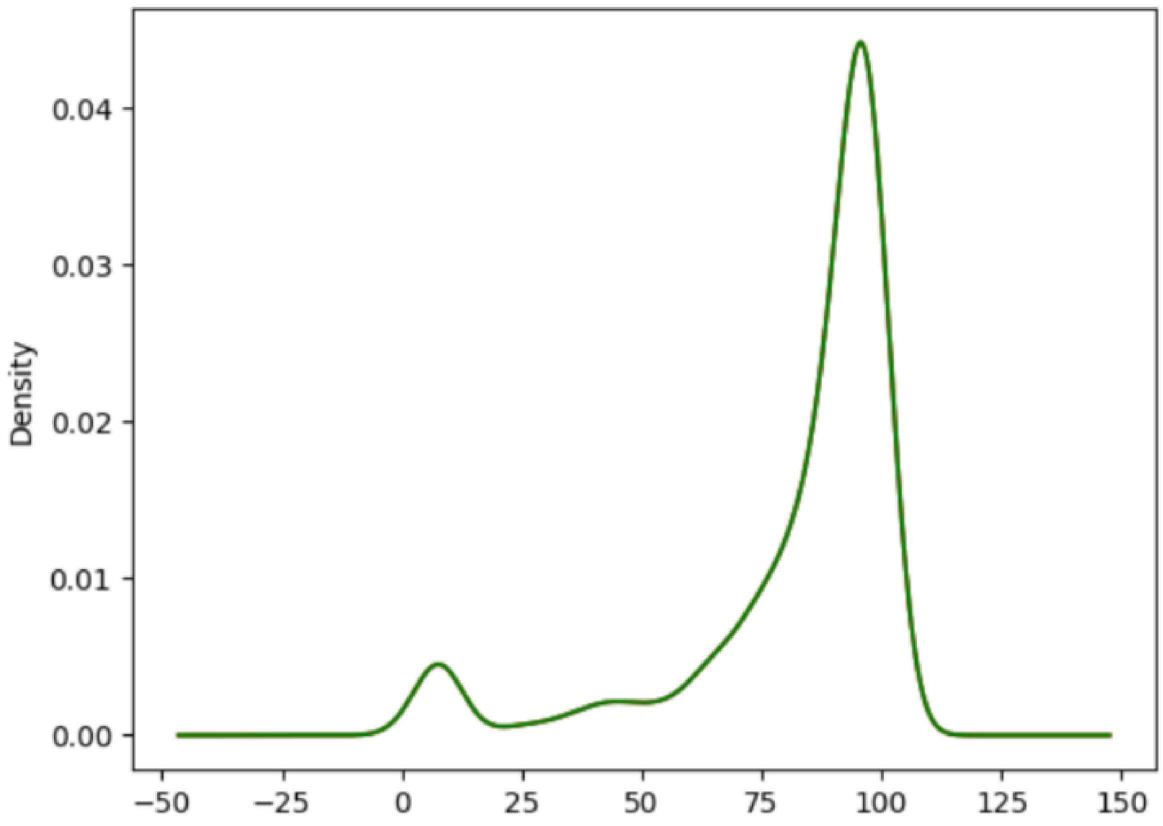


Figure:5 Density plot for diphtheria

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
original_df['thinness_10-19_years'].plot.density(color='red')

# data after cca
df['thinness_10-19_years'].plot.density(color='green')
```

```
<Axes: ylabel='Density'>
```

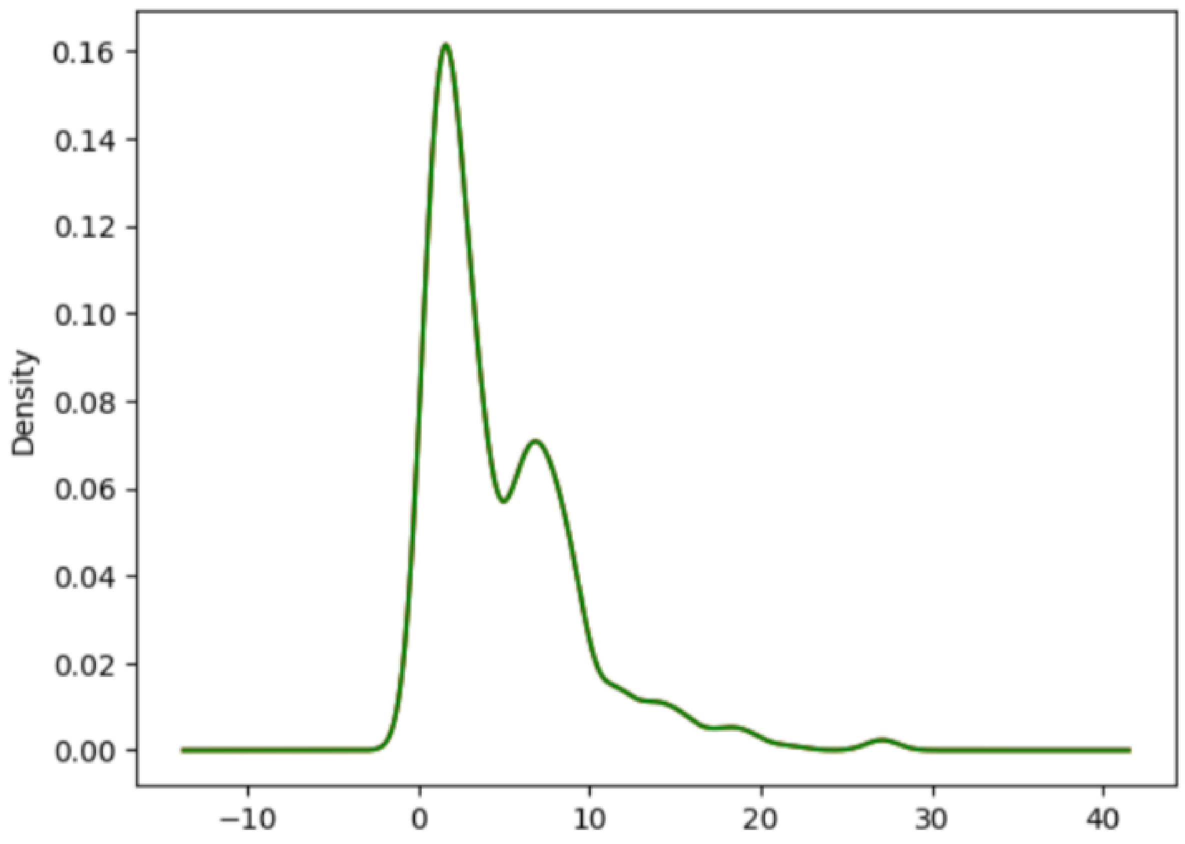


Figure:6 Density plot for thinness_10-19_years

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
original_df['thinness_5-9_years'].plot.density(color='red')

# data after cca
df['thinness_5-9_years'].plot.density(color='green')
```

```
<Axes: ylabel='Density'>
```

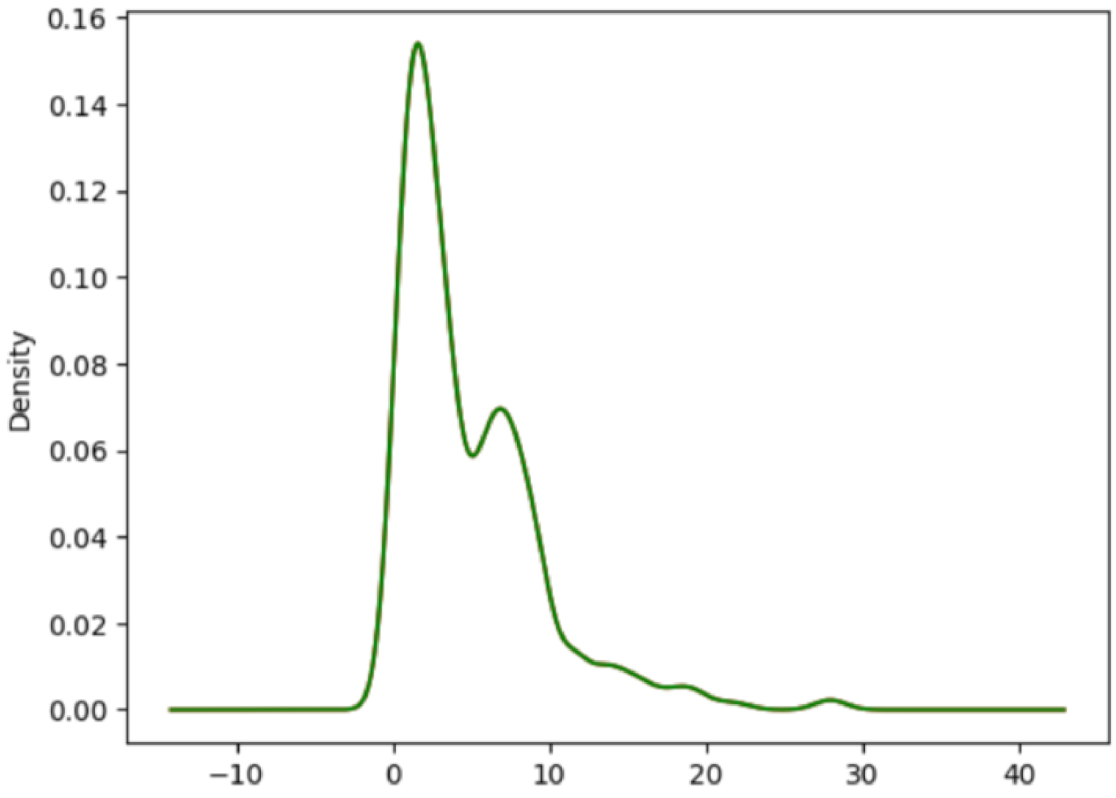


Figure:7 Density plot for thinness 5-9 years'

In all the above density plots the original data is represented in red and the data obtained after dropping some rows is represented in green. When the two density plots overlap so much that only the green data is visible, it suggests that there is little to no difference in the distribution of the data column after removing those specific rows.
This can be interpreted as an indication that the rows removed did not significantly impact the overall distribution of the data. It implies that the retained data points, represented by the green density plot, are still a good representation of the original dataset's distribution, even after some data points have been dropped.

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import pandas as pd
import numpy as np

cols_drop=['country','status']
temp_df=df.drop(cols_drop,axis=1)
cols_df=df[cols_drop]
# Create an instance of IterativeImputer
```

```
imputer = IterativeImputer(random_state=0)

# Fit and transform the imputer on your data
X_imputed = imputer.fit_transform(temp_df)

# The result is a NumPy array with imputed values
# Convert it back to a DataFrame if needed
df_imputed = pd.DataFrame(X_imputed, columns=temp_df.columns)
# new_df=pd.merge(cols_df,df_imputed,on='Year')
new_df = cols_df.merge(df_imputed, left_index=True, right_index=True)
df=new_df
```

```
nulls_breakdown(new_df)
```

```
Out of 21 total columns, 0 contain null values; 0.0% columns contain null
values.
```

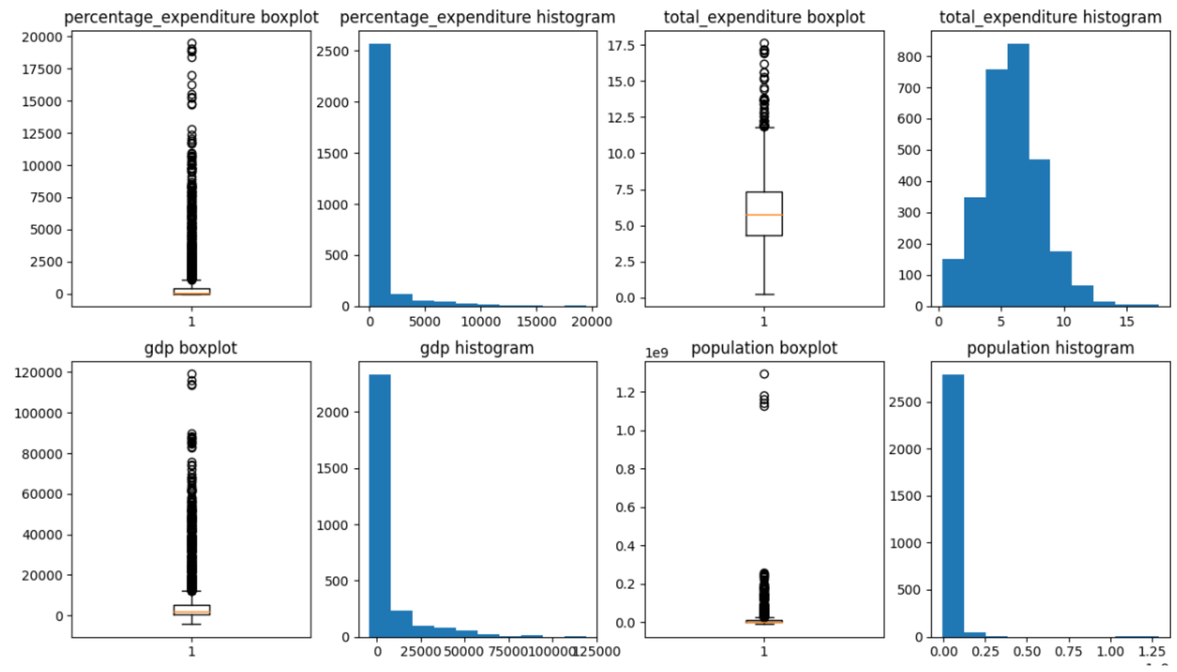
The code is used to fill in the missing values in the dataset using the IterativeImputer. The IterativeImputer works by modeling each feature with missing values as a function of other features in a round-robin fashion. This means that the imputer is able to use the information in other features to estimate the values of missing features.The result of this code is a dataset with no missing values.

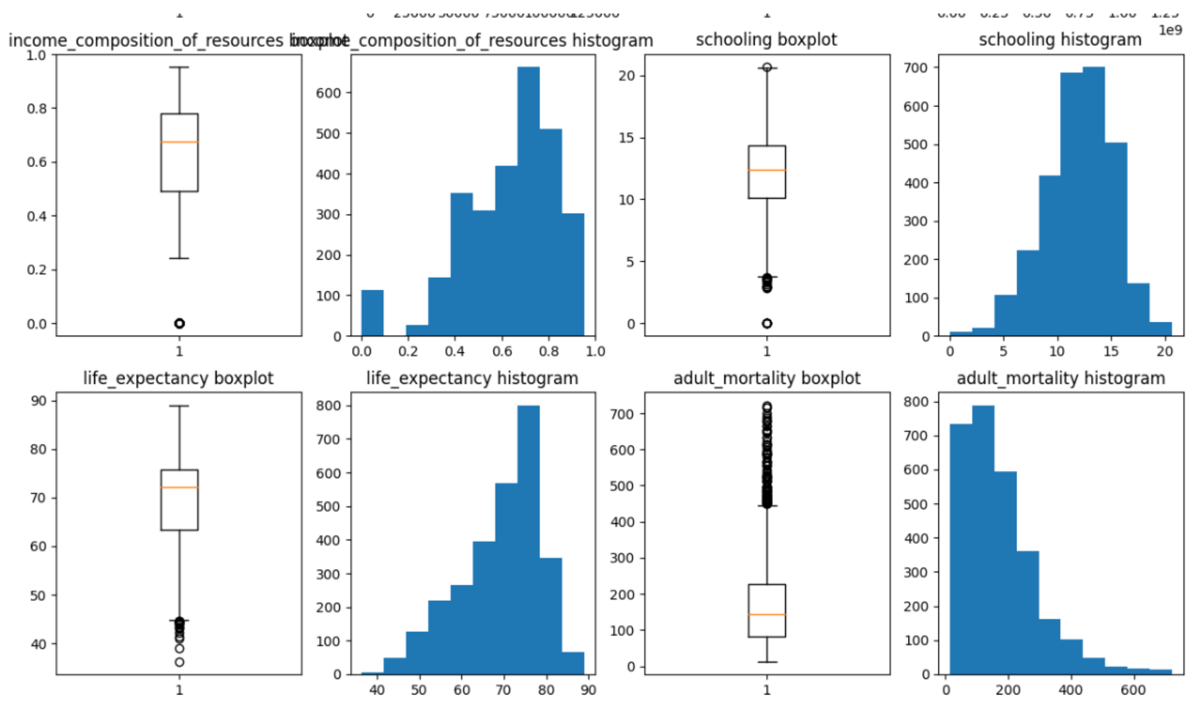
Removing outliers

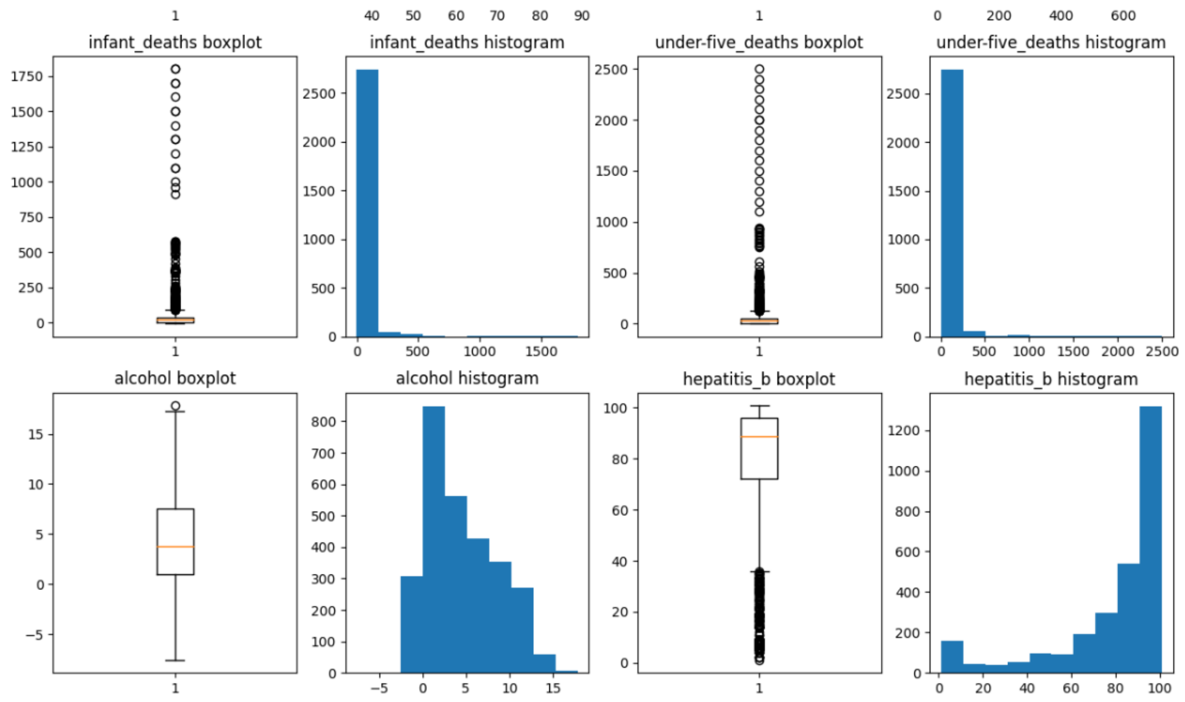
Outlier analysis is a critical step in data analysis that involves identifying and handling data points that significantly deviate from the norm. It is important for ensuring data quality, model accuracy, anomaly detection, and gaining valuable insights.

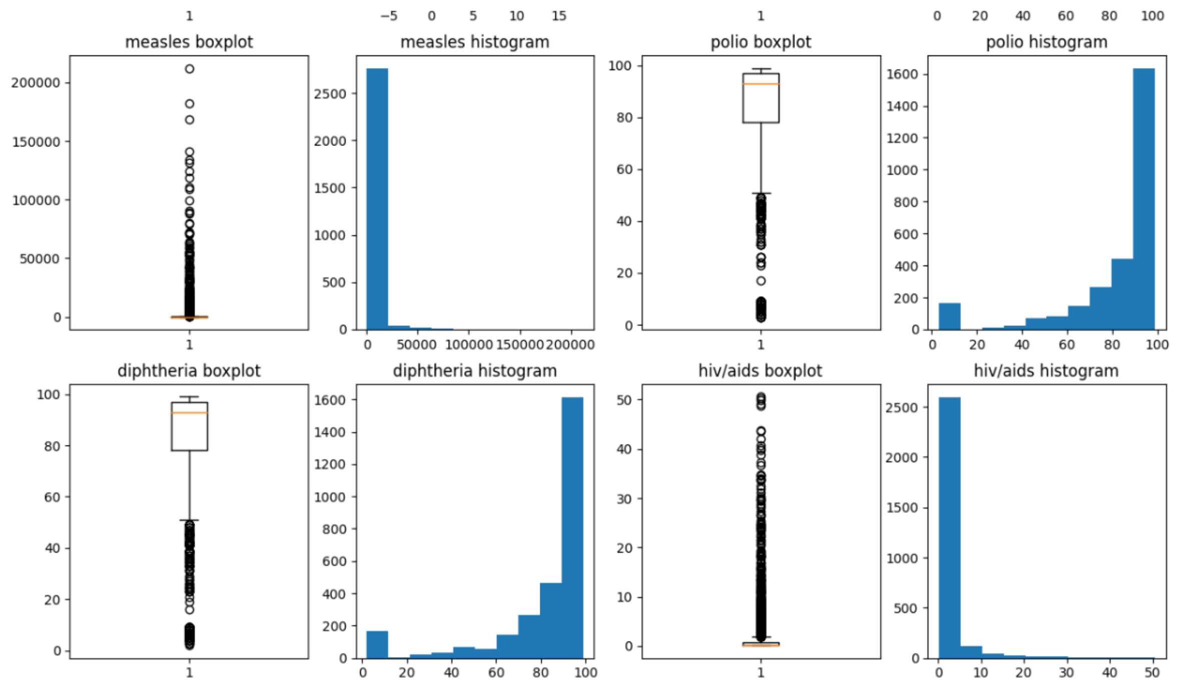
We have used box-plots along with histograms for visualizing the outlier analysis due to their effectiveness in providing a concise summary of data distributions. They offer clear visual cues for identifying outliers, showcasing quartiles and the median, making it easy to gauge central tendency and spread. Their robustness to extreme values and data distribution ensures their reliability, especially for non-normally distributed datasets. Box plots are space-efficient, allowing multiple plots to be displayed side by side for comparisons. They provide a uniform and consistent way to present data, making them a popular choice for outlier analysis and data exploration.

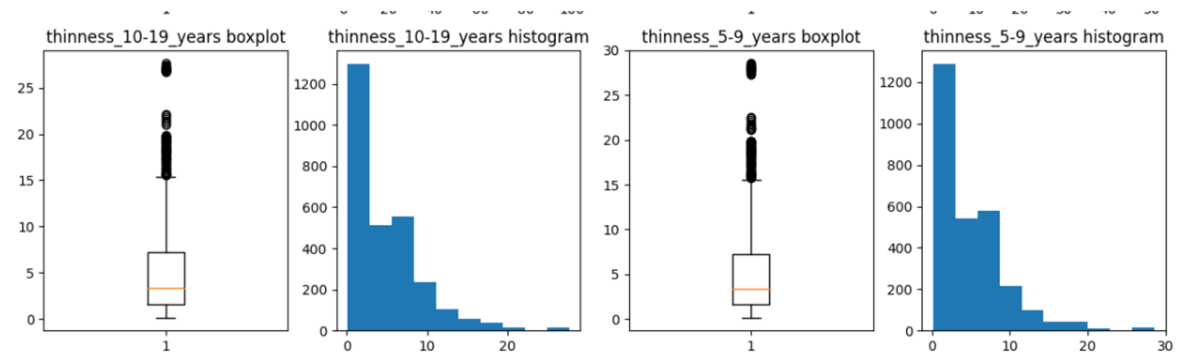
```
cont_vars = list(df.columns)[3:]
def outliers_visual(data):
    plt.figure(figsize=(15, 40))
    i = 0
    for col in cont_vars:
        i += 1
        plt.subplot(9, 4, i)
        plt.boxplot(data[col])
        plt.title('{} boxplot'.format(col))
        i += 1
        plt.subplot(9, 4, i)
        plt.hist(data[col])
        plt.title('{} histogram'.format(col))
    plt.show()
outliers_visual(df)
```




income_composition_of_resources boxplot
income_composition_of_resources histogram
schooling boxplot
schooling histogram
1e9
life_expectancy boxplot
life_expectancy histogram
adult_mortality boxplot
adult_mortality histogram


infant_deaths boxplot
infant_deaths histogram
under-five_deaths boxplot
under-five_deaths histogram
alcohol boxplot
alcohol histogram
hepatitis_b boxplot
hepatitis_b histogram


measles boxplot
measles histogram
polio boxplot
polio histogram
diphtheria boxplot
diphtheria histogram
hiv/aids boxplot
hiv/aids histogram



The outlier_count function is a tool for identifying and quantifying outliers in a specific column of a DataFrame. It accomplishes this by computing the quartiles (Q1 and Q3) and subsequently determining an outlier range based on the Interquartile Range (IQR) method. Any data points falling outside this range are considered outliers. The function then reports the count of outliers and the percentage of the data that they represent, providing valuable insights into the data's distribution and potential anomalies.

```
def outlier_count(col, data=df):
    print(15*'-' + col + 15*'-')
    q75, q25 = np.percentile(data[col], [75, 25])
    iqr = q75 - q25
    min_val = q25 - (iqr*1.5)
    max_val = q75 + (iqr*1.5)
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < 
min_val))[0])
    outlier_percent = round(outlier_count/len(data[col])*100, 2)
    print('Number of outliers: {}'.format(outlier_count))
    print('Percent of data that is outlier: {}%'.format(outlier_percent))

for col in cont_vars:
    outlier_count(col)
```

```
---------------percentage_expenditure---------------
Number of outliers: 373
Percent of data that is outlier: 13.14%
---------------total_expenditure---------------
Number of outliers: 43
Percent of data that is outlier: 1.52%
---------------gdp---------------
Number of outliers: 401
```

```
Percent of data that is outlier: 14.13%
---------------population---------------
Number of outliers: 212
Percent of data that is outlier: 7.47%
---------------income_composition_of_resources---------------
Number of outliers: 114
Percent of data that is outlier: 4.02%
---------------schooling---------------
Number of outliers: 26
Percent of data that is outlier: 0.92%
---------------life_expectancy---------------
Number of outliers: 17
Percent of data that is outlier: 0.6%
---------------adult_mortality---------------
Number of outliers: 94
Percent of data that is outlier: 3.31%
---------------infant_deaths---------------
Number of outliers: 149
Percent of data that is outlier: 5.25%
---------------under-five_deaths---------------
Number of outliers: 159
Percent of data that is outlier: 5.6%
---------------alcohol---------------
Number of outliers: 1
Percent of data that is outlier: 0.04%
---------------hepatitis_b---------------
Number of outliers: 262
Percent of data that is outlier: 9.23%
---------------measles---------------
Number of outliers: 525
Percent of data that is outlier: 18.5%
---------------polio---------------
Number of outliers: 268
Percent of data that is outlier: 9.44%
---------------diphtheria---------------
Number of outliers: 290
Percent of data that is outlier: 10.22%
---------------hiv/aids---------------
Number of outliers: 525
Percent of data that is outlier: 18.5%
---------------thinness_10-19_years---------------
Number of outliers: 88
Percent of data that is outlier: 3.1%
---------------thinness_5-9_years---------------
```

Number of outliers: 96
Percent of data that is outlier: 3.38%

To handle the outliers we have implemented the winsorization technique.
Winsorization is a method for handling outliers by transforming extreme values to be less extreme, rather than removing them from the dataset. It is a useful technique in situations where you want to maintain the integrity of your data while reducing the influence of outliers on statistical analyses.

```
def test_wins(col, lower_limit=0, upper_limit=0, show_plot=True):
    wins_data = winsorize(df[col], limits=(lower_limit, upper_limit))
    wins_dict[col] = wins_data
    if show_plot == True:
        plt.figure(figsize=(15,5))
        plt.subplot(121)
        plt.boxplot(df[col])
        plt.title('original {}'.format(col))
        plt.subplot(122)
        plt.boxplot(wins_data)
        plt.title('wins=({},{}) {}'.format(lower_limit, upper_limit, col))
        plt.show()
wins_dict = {}
test_wins(cont_vars[0], lower_limit=.01, show_plot=True)
test_wins(cont_vars[1], upper_limit=.04, show_plot=False)
test_wins(cont_vars[2], upper_limit=.05, show_plot=False)
test_wins(cont_vars[3], upper_limit=.0025, show_plot=False)
test_wins(cont_vars[4], upper_limit=.135, show_plot=False)
test_wins(cont_vars[5], lower_limit=.1, show_plot=False)
test_wins(cont_vars[6], upper_limit=.19, show_plot=False)
test_wins(cont_vars[7], upper_limit=.05, show_plot=False)
test_wins(cont_vars[8], lower_limit=.1, show_plot=False)
test_wins(cont_vars[9], upper_limit=.02, show_plot=False)
test_wins(cont_vars[10], lower_limit=.105, show_plot=False)
test_wins(cont_vars[11], upper_limit=.185, show_plot=False)
test_wins(cont_vars[12], upper_limit=.105, show_plot=False)
test_wins(cont_vars[13], upper_limit=.07, show_plot=False)
test_wins(cont_vars[14], upper_limit=.035, show_plot=False)
test_wins(cont_vars[15], upper_limit=.035, show_plot=False)
test_wins(cont_vars[16], lower_limit=.05, show_plot=False)
test_wins(cont_vars[17], lower_limit=.025, upper_limit=.005,
show_plot=False)
```

Regression analysis is a potent statistical tool for quantifying the relationships between different factors and life expectancy. It provides a structured method for predicting and comprehending the influence of these factors, making it indispensable for making informed decisions in the realms of public health, policy, healthcare, and research.
The code starts by preparing the dataset and defining the features (independent variables) and the target variable (life expectancy). It then splits the dataset into training and testing sets, with 70% of the data used for training and 30% for testing.
Next, the code initializes a Lasso regression model. Lasso is a linear regression technique that performs both feature selection and prediction. It does this by adding a regularization term (controlled by the alpha parameter) that encourages certain feature coefficients to be exactly zero. As a result, Lasso can automatically select the most relevant features while predicting the target variable (life expectancy in this case).
The code fits the Lasso model to the training data, meaning it adjusts the model's parameters to minimize the prediction error based on the training data. Once the model is trained, it's used to make predictions on the test data.
The code then assesses the model's performance using two metrics:
1.Mean Squared Error (MSE): This metric quantifies the average squared difference between the model's predictions and the actual life expectancy values in the test set. A lower MSE indicates a better-performing model.
2.R-squared (R2): R2 measures the proportion of the variance in the target variable that is predictable from the independent variables. It ranges from 0 to 1, with higher values indicating a better fit.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score


cols=wins_df.columns.tolist()
cols.remove('life_expectancy')
cols.remove('country')
cols.remove('status')
cols.remove('year')
```

```
# Select independent variables (X) and the target variable (y)
X = wins_df[cols]  # Include your independent variables
y = wins_df['life_expectancy']  # Your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Lasso regression model
alpha = 0.01  # Adjust the regularization strength (alpha) as needed
model = Lasso(alpha=alpha)
model.fit(X_train, y_train)

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

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Print the coefficients of the selected features
feature_names = X.columns
coefficients = model.coef_
```

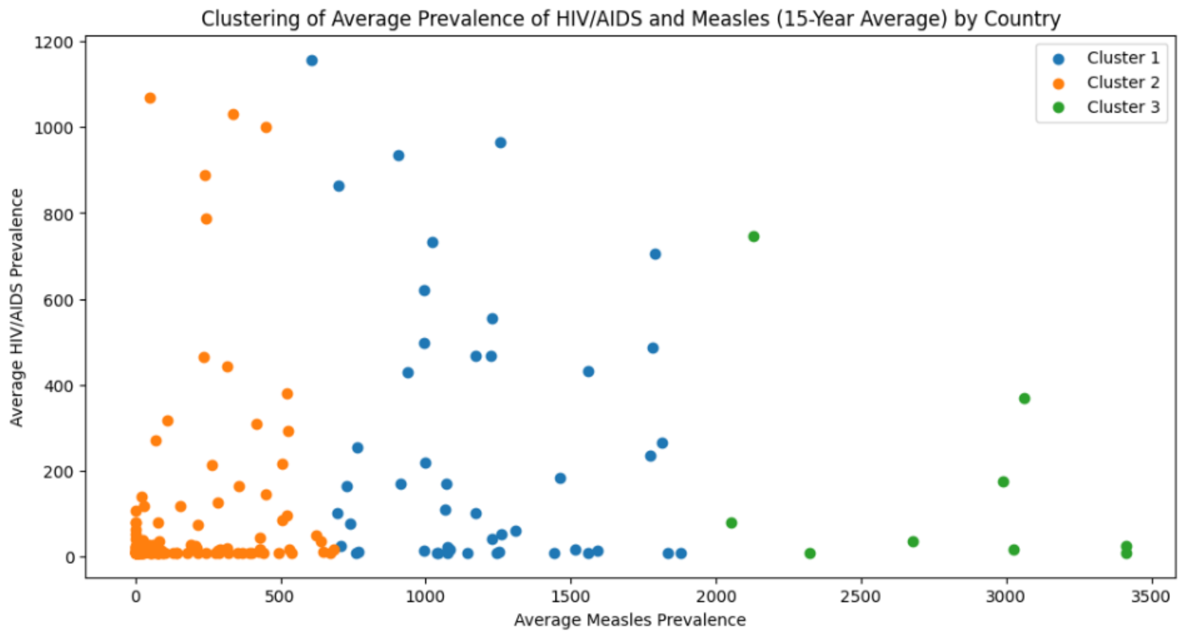
```
Mean Squared Error: 12.918989842837451 R-squared: 0.8355016215230893
```

Here,the MSE of 12.92 suggests that, on average, the model's predictions are off by approximately 12.92 units in terms of life expectancy. The R^2 value of 0.836 indicates that the model is able to explain around 83.6% of the variance in life expectancy based on the selected independent variables. These metrics collectively show that the model is a good fit for the data and is capable of explaining a significant portion of the variation in life expectancy.

We have performed K-Means clustering on the average prevalence data for measles and HIV/AIDS for different countries and visualizes the clusters in a scatter plot, allowing you to see how the countries group together based on these two health-related features.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from sklearn.cluster import KMeans

# Define the features for clustering
features = ['measles', 'hiv/aids']

# Group the data by country and calculate the average prevalence over 15
years
average_data = wins_df.groupby('country')[features].mean()

# Create a K-Means model with the desired number of clusters
n_clusters = 3  # You can adjust the number of clusters
kmeans = KMeans(3, random_state=0)
cluster_labels = kmeans.fit_predict(average_data)

# Plot the clustered data
plt.figure(figsize=(12, 6))
for i in range(n_clusters):
    plt.scatter(
        average_data[cluster_labels == i]['measles'],
        average_data[cluster_labels == i]['hiv/aids'],
        label=f'Cluster {i + 1}',
    )

# Set labels and legend
plt.xlabel('Average Measles Prevalence')
plt.ylabel('Average HIV/AIDS Prevalence')
plt.title('Clustering of Average Prevalence of HIV/AIDS and Measles
(15-Year Average) by Country')
plt.legend()

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



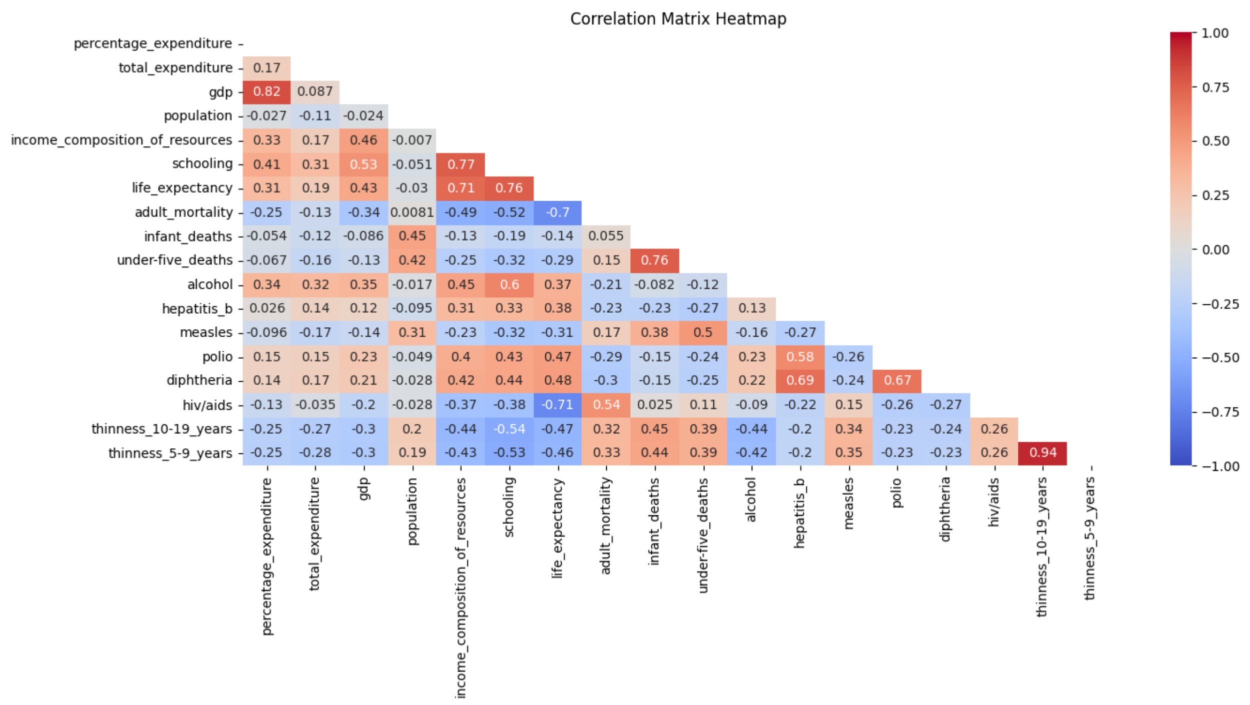
The graph resulting from K-Means clustering shows how countries group based on their average measles and HIV/AIDS prevalence. It identifies clusters of countries with similar disease patterns, aiding in the comparison of health outcomes and the targeting of interventions. Outliers can be spotted, and the analysis enables data-driven decision-making, resource allocation, and further research into common risk factors or regional trends in public health. The visualization simplifies data analysis and informs policy and public health strategies.

Applying user-defined filters on cleaned dataset

Filtering technique to find countries with the life expectancy greater than most as well as the countries with life expectancy lower than most.

Starting with showing the correlation matrix between all variables and visualizing them.

```
mask = np.triu(wins_df[cont_vars].corr())
plt.figure(figsize=(15,6))
sns.heatmap(wins_df[cont_vars].corr(), annot=True, fmt='.2g', vmin=-1,
vmax=1, center=0, cmap='coolwarm', mask=mask)
plt.ylim(18, 0)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



This matrix gives us the attributes that correlate with each other most.
Now we find out the top 4 most correlated columns with 'life_expectancy' using the above correlation matrix. This is essential for knowing which features we are to select for comparison of ‘life expectancies’ among different countries.

```python
correlation_matrix=wins_df[cont_vars].corr()
correlation_with_life =
correlation_matrix['life_expectancy'].abs().sort_values(ascending=False)

# Exclude 'lifeexpectancy' itself from the list
correlation_with_life = correlation_with_life.drop('life_expectancy')

# Get the top four correlated columns
top_correlations = correlation_with_life.head(4)

# Extract the column names
top_correlation_columns = top_correlations.index.tolist()

print("Top four correlated columns with 'life_expectancy':")
print(top_correlation_columns)
```

```
Top four correlated columns with 'life_expectancy':
['schooling', 'income_composition_of_resources', 'hiv/aids', 'adult_mortality']
```

Creating a new dataframe using the top 4 correlated columns along with columns 'year' and 'life_expectancy'. So that data is easily accessed.

```
colsel = top_correlation_columns + ['year'] + ['life_expectancy']
new_df = df_filtered[colsel]
new_df
```

	schooling	income_composition_of_resources	hiv/aids	adult_mortality	year	life_expectancy
0	10.1	0.479	0.1	263.0	2015.0	65.0
16	14.2	0.762	0.1	74.0	2015.0	77.1
32	14.4	0.743	0.1	19.0	2015.0	75.6
48	11.4	0.531	1.9	335.0	2015.0	52.4
64	13.9	0.784	0.2	13.0	2015.0	76.4
...	...	...	...	...	...	...
2808	14.3	0.769	0.1	157.0	2015.0	74.1
2824	12.6	0.678	0.1	127.0	2015.0	76.0
2840	9.0	0.499	0.1	224.0	2015.0	65.7
2856	12.5	0.576	4.1	33.0	2015.0	61.8
2872	10.3	0.507	6.2	336.0	2015.0	67.0

177 rows × 6 columns

Now, using the new data frame we wanted to find out the countries within the highest range of life expectancy and the lowest range of life expectancy. For that we first calculated the first and the third quartile value for the top 4 selected columns and also for the life_expectancy column.

Finding the quartile range to write the high & low filters

```
for column in top_correlation_columns:
    q1 = np.percentile(new_df[column], 25)
    q3 = np.percentile(new_df[column], 75)
    iqr = q3 - q1
    print(f"q1 for {column}: {q1}")
    print(f"q3 for {column}: {q3}")
```

```
q1 for schooling: 10.8
q3 for schooling: 15.0
q1 for income_composition_of_resources: 0.558
q3 for income_composition_of_resources: 0.804
q1 for hiv/aids: 0.1
q3 for hiv/aids: 0.4
q1 for adult_mortality: 75.0
q3 for adult_mortality: 211.0
```

```
q1=np.percentile(new_df['life_expectancy'],25)
q3=np.percentile(new_df['life_expectancy'],75)
print(q1)
print(q3)
```

```
65.8
76.9
```

Then after calculating the quartile values, we filtered out the countries with higher life_expectancy values(above 3rd quartile value) and the countries with lower life_expectancy values (below 1st quartile value) on the basis of positive and negative correlation with life_expectancy of the top 4 selected columns.

Filtering countries with high life expectancy

```
# Filter out countries based on correlations
filtered_countries_high = wins_df[
    ((wins_df['adult_mortality'] < 75.5)  |
    (wins_df['hiv/aids'] < 0.1)  |
    (wins_df['income_composition_of_resources'] > 0.7945)  |
    (wins_df['schooling'] > 14.85))  &
    (wins_df['life_expectancy'] > 76.95)
     & (wins_df['year'] == 2015)
]

# Output the filtered countries
print(filtered_countries_high.sort_values('life_expectancy'))

# Extract the 'country' column
countries = df['country'].tolist()

# Print the list of country names
```

```
print(countries)
```

['United Arab Emirates', 'Lithuania', 'Malaysia', 'Mali', 'Nepal', 'Netherlands', 'Nigeria', 'Philippines', 'Jamaica', 'Poland', 'Qatar', 'Sierra Leone', 'Slovakia', 'South Africa', 'Suriname', 'Turkey', 'Turkmenistan', 'Portugal', 'Ukraine', 'Albania', 'Iraq', 'Australia', 'Austria', 'Belgium', 'Brunei Darussalam', 'Canada', 'Chile', 'Costa Rica', 'Ireland', 'Cuba', 'Democratic Republic of the Congo', 'Eritrea', 'Fiji', 'Finland', 'Georgia', 'Ghana', 'Hungary', 'Cyprus', 'Israel']

These are the countries with higher life expectancy.

Filtering countries with low life expectancy

Utilized quartile values to filter countries based on life expectancy and correlated variables.

```
# Filter out countries based on correlations
filtered_countries_low = wins_df[
    ((wins_df['adult_mortality'] > 213.0) |
    (wins_df['hiv/aids'] > 0.4) |
    (wins_df['income_composition_of_resources'] < 0.575) |
    (wins_df['schooling'] < 11.1)) &
    (wins_df['life_expectancy'] < 65.75)
     & (wins_df['year'] == 2015)
]

# Output the filtered countries
print(filtered_countries_low.sort_values('life_expectancy'))
```

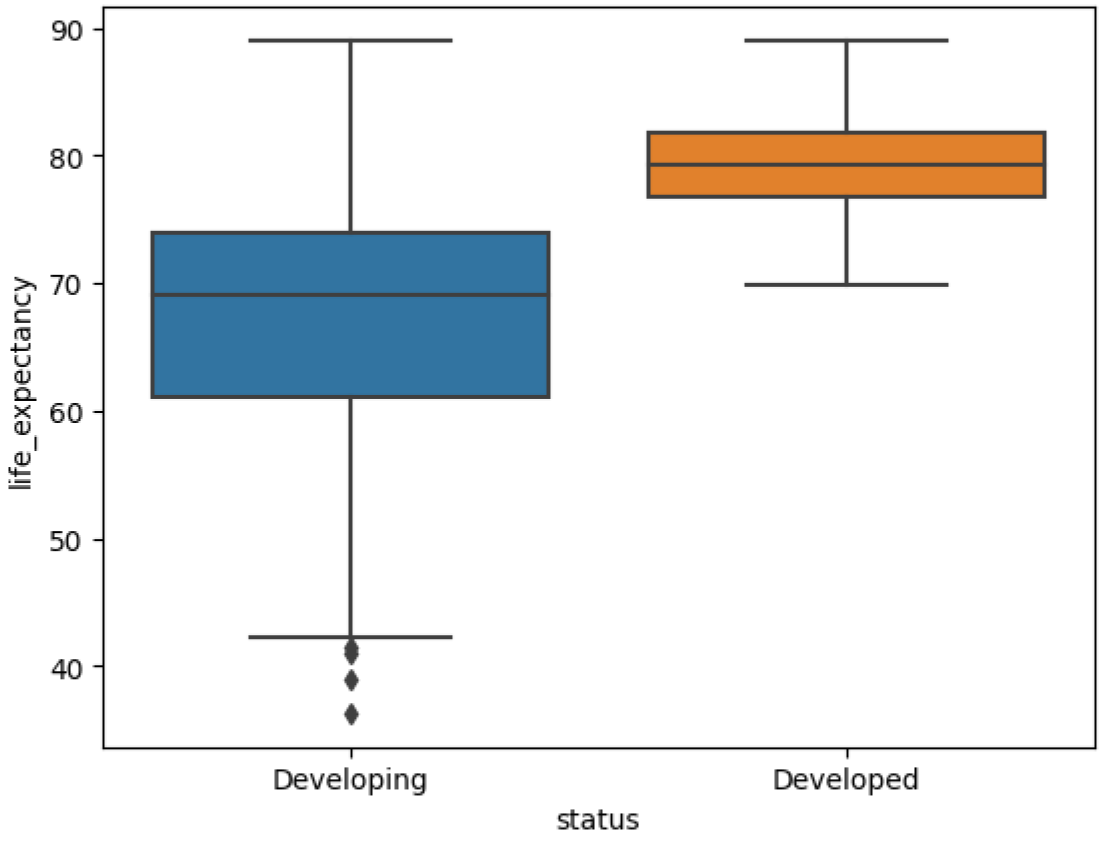
['Seychelles', 'Angola', 'Central African Republic', 'Chad', "Côte d'Ivoire", 'Lebanon', 'Niger', 'Solomon Islands', 'Cameroon', 'Morocco', 'Maldives', 'El Salvador', 'Madagascar', 'Guinea', 'Sri Lanka', 'Guatemala', 'Burundi', "Democratic People's Republic of Korea", 'Tajikistan', 'Burkina Faso', 'Benin', 'Gabon', 'Lesotho', 'Vanuatu', 'Nicaragua', 'Uganda', 'Trinidad and Tobago', 'Germany', 'Panama', 'Somalia', 'Malta', 'Kazakhstan', 'Guyana', 'Denmark', 'Comoros', 'Swaziland', 'Equatorial Guinea', 'Congo', 'Estonia', 'Afghanistan', 'Luxembourg', 'Kyrgyzstan', 'Uzbekistan', 'Botswana']

These countries are having lower life expectancy.

Basic plots

1. Status VS Life expectancy: The accompanying boxplot illustrates a compelling comparison of life expectancy across different countries categorized by their status, either developed or developing. This visual representation vividly showcases the notable disparity in life expectancy between these two groups. Developed countries, characterized by advanced healthcare systems and stable economies, exhibit significantly higher life expectancy compared to developing nations. This disparity highlights the critical importance of addressing healthcare infrastructure, education, and economic stability in developing countries to bridge the gap and ensure a healthier future for all.

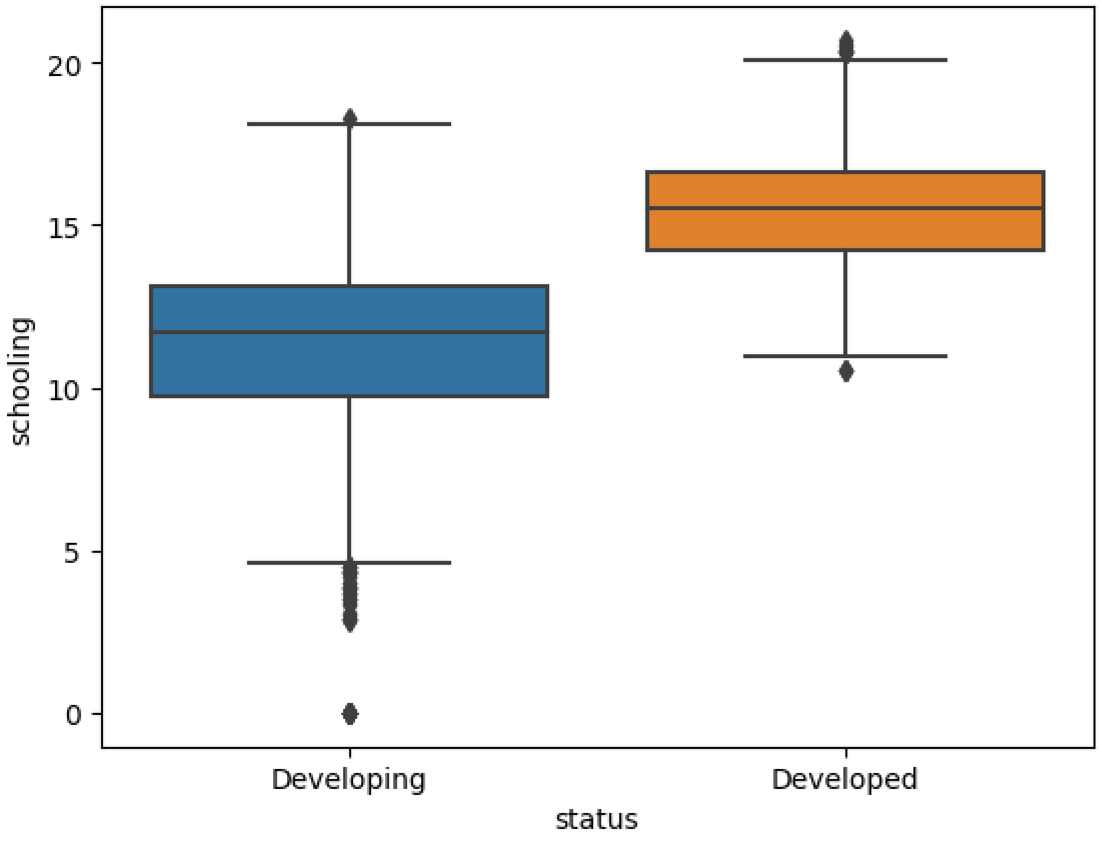
```
# boxplot for Status and life expectancy
sns.boxplot(x='status', y='life_expectancy', data=df)
plt.show()
```



2. Status vs Schooling: Examining the relationship between schooling and development, this boxplot offers a striking visualization of the disparities in educational access and infrastructure between different countries. Developed nations, characterized by

advanced educational systems and economic stability, demonstrate higher levels of schooling on average. In contrast, developing countries face challenges in providing widespread access to quality education. This disparity underscores the critical importance of investments in educational infrastructure, access to quality schooling, and economic development in less affluent regions. Addressing these disparities in education is a key step towards fostering equitable global development and ensuring a brighter future for all.

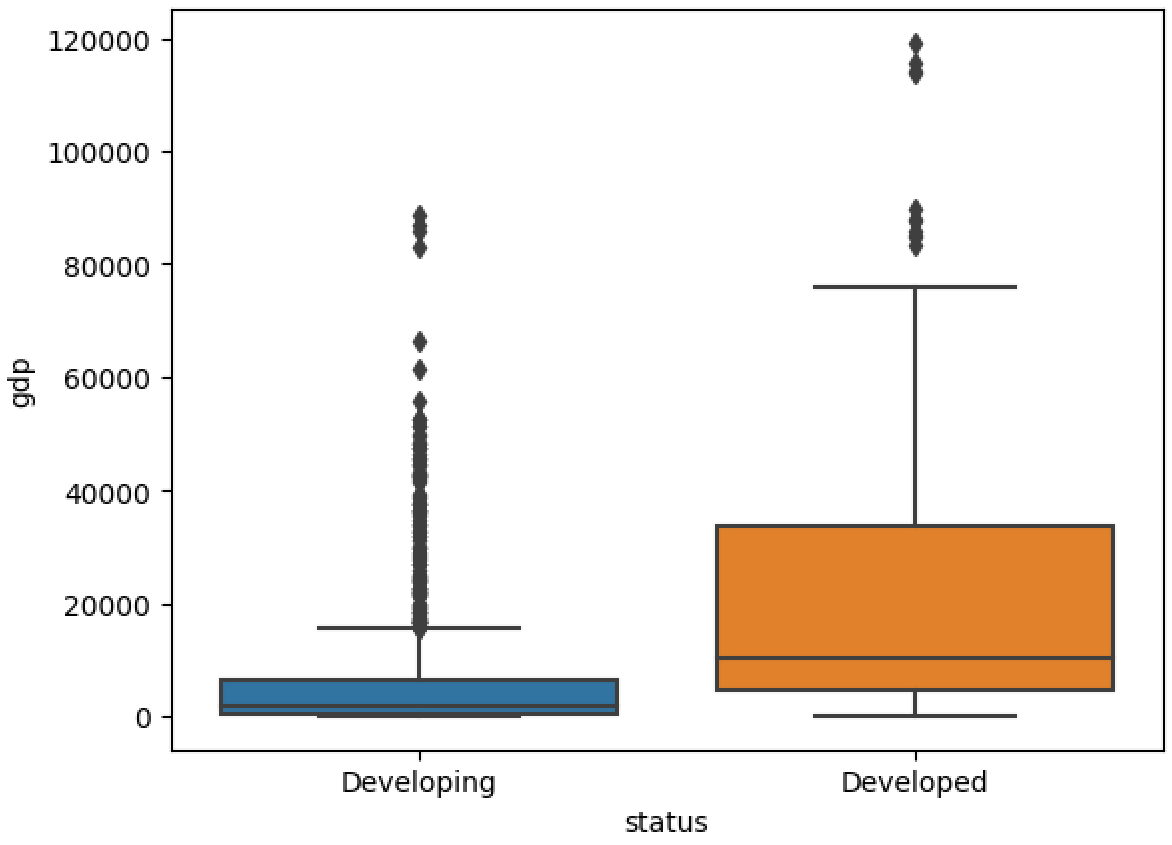
```
# boxplot for Status and schooling
sns.boxplot(x='status', y='schooling', data=df)
plt.show()
```



3. Status vs GDP: The accompanying boxplot provides a compelling comparison between the economic statuses of countries, categorized as either developed or developing, and their respective Gross Domestic Product (GDP). Developed nations, characterized by advanced industries, innovation, and stable economies, exhibit substantially higher GDP figures when compared to their developing counterparts. This disparity underscores the economic challenges faced by developing countries and emphasizes the need for targeted strategies to promote economic growth, investment, and technological

advancement. Addressing these disparities in GDP is fundamental to achieving global economic balance and fostering sustainable development worldwide.

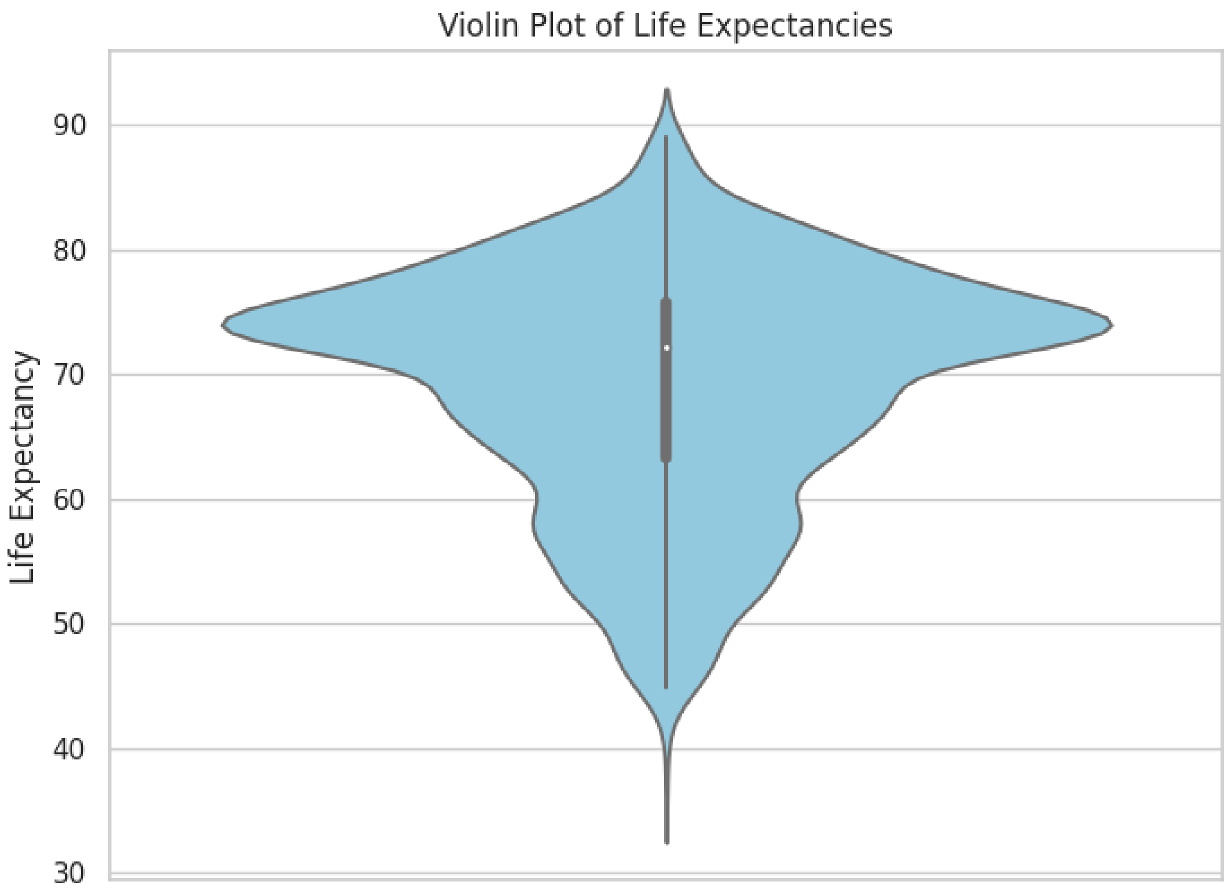
```
# boxplot for Status and gdp
sns.boxplot(x='status', y='gdp', data=df)
plt.show()
```



4. Violin plot for life expectancy

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a violin plot using seaborn
sns.set(style="whitegrid")  # Set the style of the plot
plt.figure(figsize=(8, 6))  # Set the size of the plot

# Create the violin plot
sns.violinplot(y=df['life_expectancy'], color="skyblue")
```

```
# Add labels and title
plt.ylabel("Life Expectancy")
plt.title("Violin Plot of Life Expectancies")

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



Interactive plots

Combining Attributes in a 3D Interactive Graph: Traditional 2D graphs provide valuable insights, but they often miss the complexity of real-world relationships. By incorporating multiple attributes into a 3D interactive graph, we gain a nuanced perspective on the interplay between a country's status (developed or developing), life expectancy, schooling, and GDP. This multidimensional visualization allows us to explore the intricate connections among these factors. Developed nations stand out with higher life expectancy, superior schooling, and robust GDP, highlighting the synergy between economic prosperity, education, and health. Countries with higher GDP and schooling, particularly those that are developed, exhibit significantly larger life expectancy, underscoring the pivotal role of education and economic stability in fostering longer, healthier lives. This interactive graph not only captures the disparities but also serves as a powerful tool for policymakers and researchers, illuminating the complex dynamics that shape global development and offering valuable insights for creating more equitable and prosperous societies worldwide.

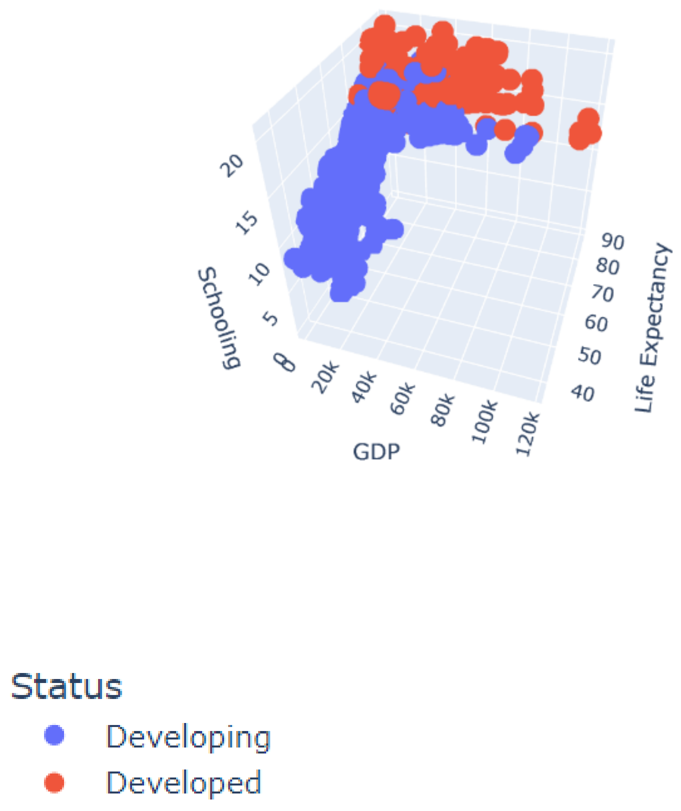
```
!pip install plotly
import plotly.express as px
```

```
import plotly.express as px

# Sample data (replace this with your actual dataset)
countries = df['country']
GDP = df['gdp']
life_expectancy = df['life_expectancy']
schooling = df['schooling']
status = df['status']

# Create a DataFrame
import pandas as pd
data = pd.DataFrame({'Country': countries, 'GDP': GDP, 'Life Expectancy':
life_expectancy, 'Schooling': schooling, 'Status': status})

# Create an interactive 3D scatter plot using Plotly
fig = px.scatter_3d(data, x='GDP', y='Life Expectancy', z='Schooling',
color='Status',
                    labels={'x': 'GDP', 'y': 'Life Expectancy', 'z':
'Schooling'},
                    title='Interactive 3D Graph: Life Expectancy based on
status, schooling, and GDP')

# Customize hover text
fig.update_traces(textposition='top center')

# Show the interactive plot
fig.show()
```

The above code uses the Plotly library to create an interactive 3D scatter plot representing the relationships among life expectancy, GDP, schooling, and the status (developed or developing) of various countries. The process begins by importing the necessary libraries: Plotly for interactive visualization and pandas for data manipulation. The data, presumably loaded from a

DataFrame named df, is structured into arrays for countries, GDP, life expectancy, schooling, and status. These arrays are then combined into a pandas DataFrame. Plotly Express (px) is employed to generate the 3D scatter plot. The scatter_3d function is used to define the x, y, and z axes as GDP, life expectancy, and schooling respectively, while the color parameter distinguishes between developed and developing countries. Labels are set for the axes, and a descriptive title is provided for clarity. Additionally, the code customizes the hover text position for improved visibility. Finally, the interactive plot is displayed using the fig.show() method. This code creates a dynamic visualization, allowing users to explore the intricate interconnections between these variables and the status of countries. The resulting interactive 3D graph provides a comprehensive perspective, aiding in the analysis of global socioeconomic trends.

Interactive 3D Graph: Life Expectancy based on status, schooling, and GDP

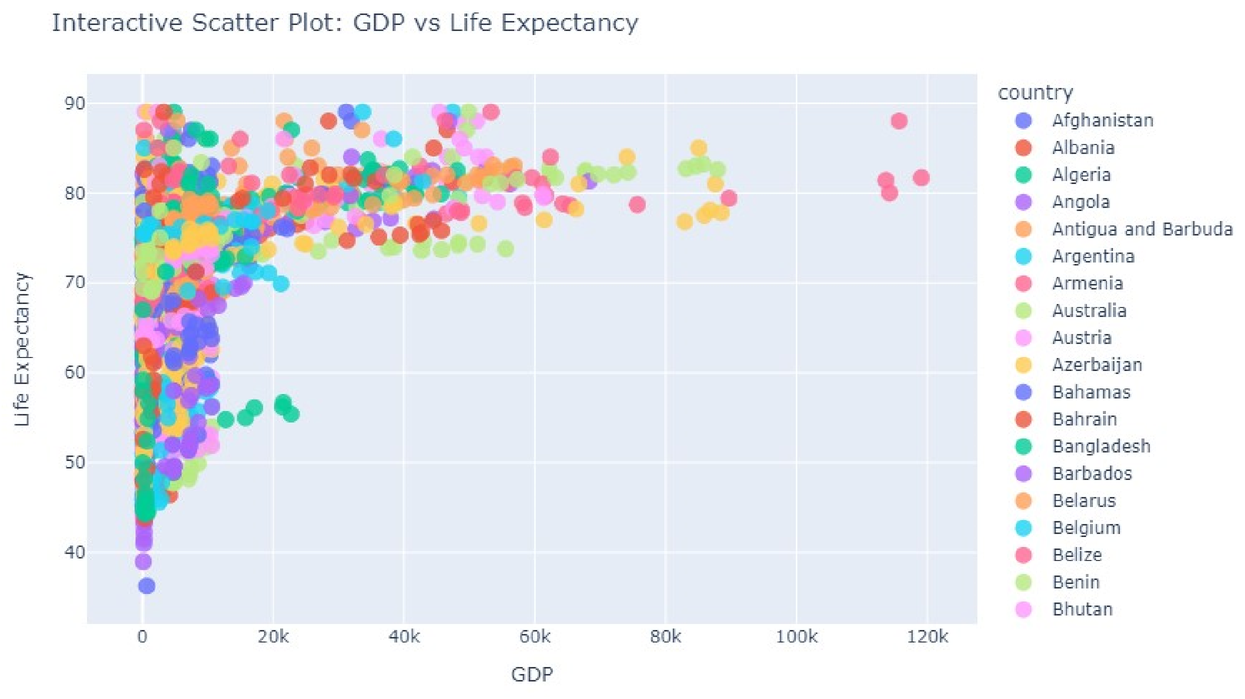


Exploring Global Trends in GDP and Life Expectancy

Introduction:
In this report, we analyze the relationship between GDP and life expectancy across various countries. We use interactive scatter plots and histograms to visualize and understand patterns and trends. The dataset includes information from multiple countries over different years.

1. Interactive Scatter Plot: GDP vs Life Expectancy

```
# Interactive Scatter Plot
scatter_plot = px.scatter(df, x='gdp', y='life_expectancy',
color='country', hover_data=['year'])
scatter_plot.update_traces(marker=dict(size=12, opacity=0.8))
scatter_plot.update_layout(title='Interactive Scatter Plot: GDP vs Life
Expectancy',
                          xaxis_title='GDP',
                          yaxis_title='Life Expectancy')

scatter_plot.show()
```



The interactive scatter plot below displays the relationship between a country's GDP and its life expectancy. Each point represents a specific country in a particular year. The x-axis represents GDP, the y-axis represents life expectancy, and the color distinguishes different countries. Hovering over points provides additional details such as the year and exact values.

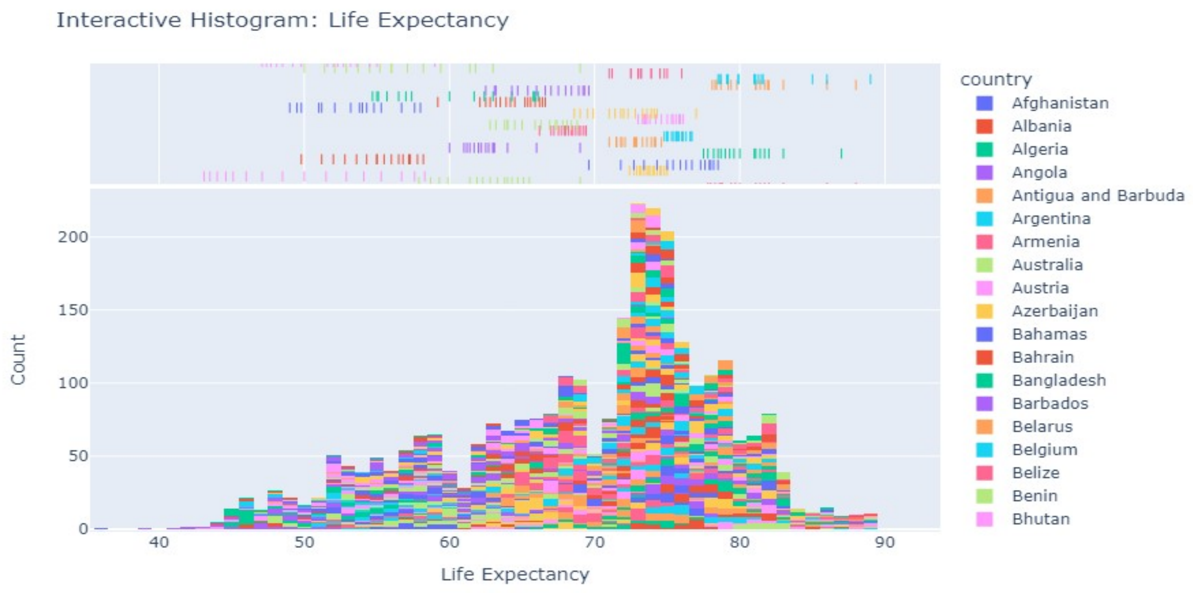
Interactive Scatter Plot: GDP vs Life Expectancy

Observations:

Countries with higher GDP tend to have a higher life expectancy.

There is a positive correlation between GDP and life expectancy.
Some outliers exist, indicating countries with unexpectedly high or low life expectancy given their GDP.

```
# Interactive Histogram
histogram = px.histogram(df, x='life_expectancy', color='country',
marginal='rug')
histogram.update_layout(title='Interactive Histogram: Life Expectancy',
                        xaxis_title='Life Expectancy',
                        yaxis_title='Count')

histogram.show()
```



2. Interactive Histogram: Life Expectancy
The interactive histogram below provides a distribution of life expectancy across the selected countries. Each bar represents a range of life expectancy values, and the color indicates different countries. Rug plots display individual data points on the x-axis.

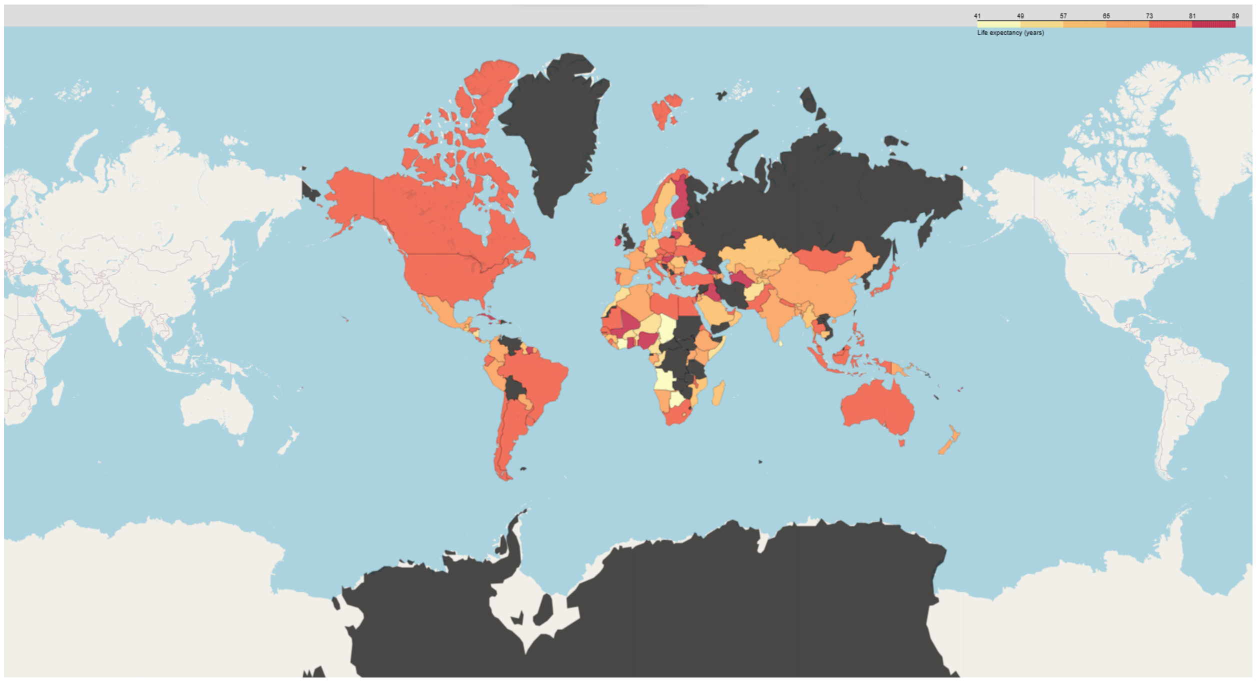
Interactive Histogram: Life Expectancy

Observations:

The majority of countries have a life expectancy between a certain range, indicating a commonality in life expectancy values.
Some countries have significantly higher or lower life expectancy compared to others, as shown by the outliers in the histogram.

Conclusion:
The interactive visualizations presented in this report help us understand the relationship between GDP and life expectancy. Higher GDP generally correlates with a higher life expectancy, suggesting that economic prosperity often leads to improved healthcare, living conditions, and overall well-being. However, it is important to note that there are exceptions and variations among countries. Further analysis could delve into the specific factors contributing to the outliers and explore the socio-economic and healthcare policies that influence life expectancy.

3.Using the geopandas and folium libraries to create an interactive choropleth map to visualize the variation in life expectancy across different countries.



Country:
India
Life
Expectancy:
69 years

We utilized the folium library to create an interactive map displaying the life expectancy data of four countries: USA, Canada, India, and China. The code constructs markers for each country, pinpointing their respective locations on the map. The markers are enriched with pop-up information, indicating the country's name and its corresponding life expectancy in years. By running this code in a Google Colab notebook, the interactive map is seamlessly displayed within the notebook interface, allowing for an engaging visual representation of life expectancy data for these countries. This approach provides a dynamic and informative way to explore geographic and demographic information, making data analysis and visualization accessible and intuitive.

<u>Country-wise Analysis</u>

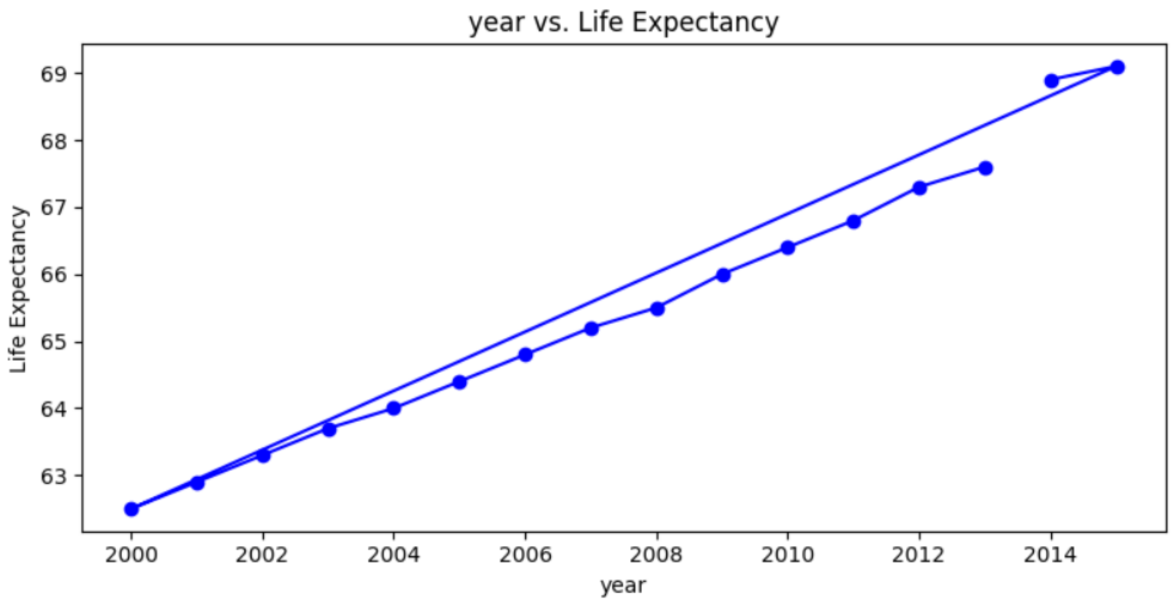
The following code will help the user to see the analysis - relationship between various factors and life expectancy, for any particular country that he wishes to see.Graphs are used for better visualization.

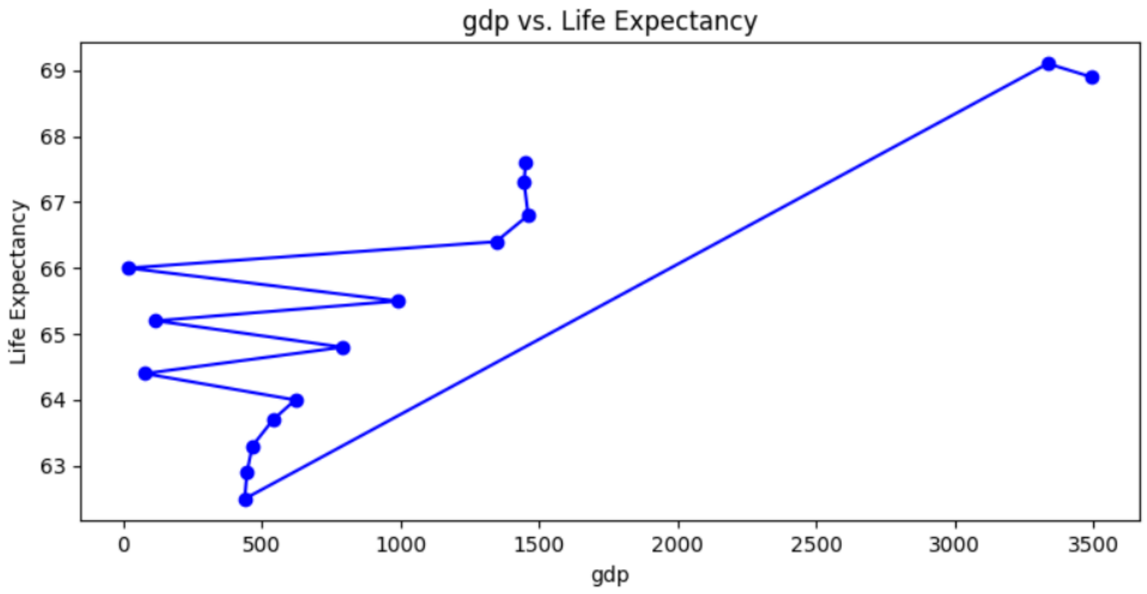
```
inp_country=input("Enter Country : ")

# Filter the DataFrame to get data for the user input country
country_df = wins_df[wins_df['country'] == inp_country]
country_df
attributes = [col for col in country_df.columns if col !=
'life_expectancy']
attributes.remove('country')
attributes.remove('status')
```

```
# Set up the number of rows and columns for subplots
num_rows = len(attributes)
num_cols = 1  # You can change this if you want a different number of
columns

# Create a new figure
fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 4*num_rows))

# Ensure axes is a 2D array
if num_rows == 1:
    axes = [axes]

# Loop through the attributes and create plots
for i, attribute in enumerate(attributes):
    ax = axes[i]

    # Scatter plot of the current attribute vs. LifeExpectancy
    ax.plot(country_df[attribute], country_df['life_expectancy'],
marker='o', linestyle='-', color='b')

    ax.set_xlabel(attribute)
    ax.set_ylabel('Life Expectancy')
    ax.set_title(f'{attribute} vs. Life Expectancy')

# Adjust spacing between subplots
plt.tight_layout()

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

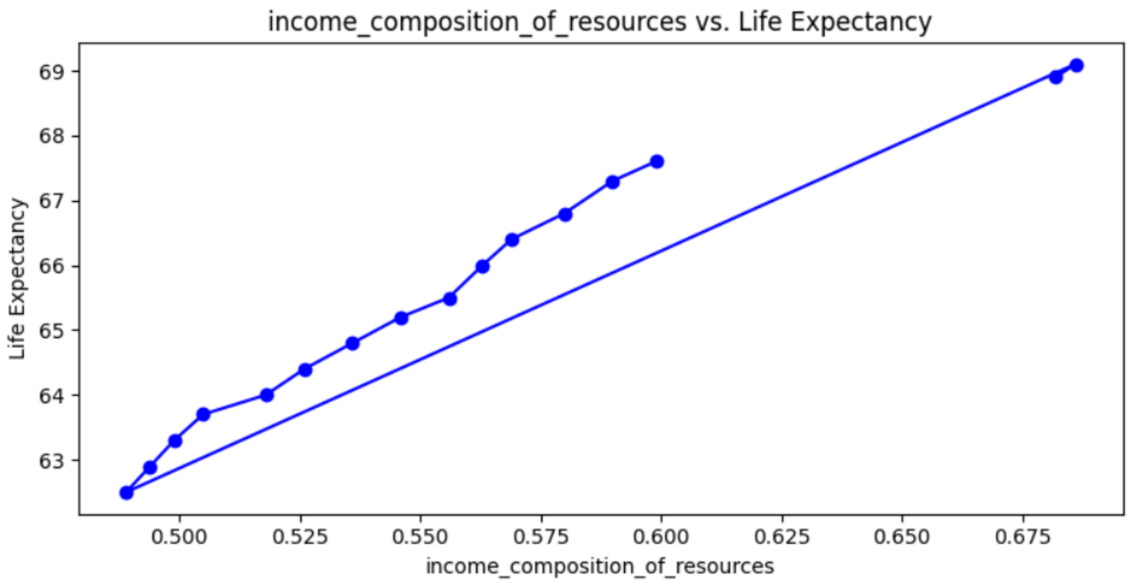
Some of the plots for India are shown below:

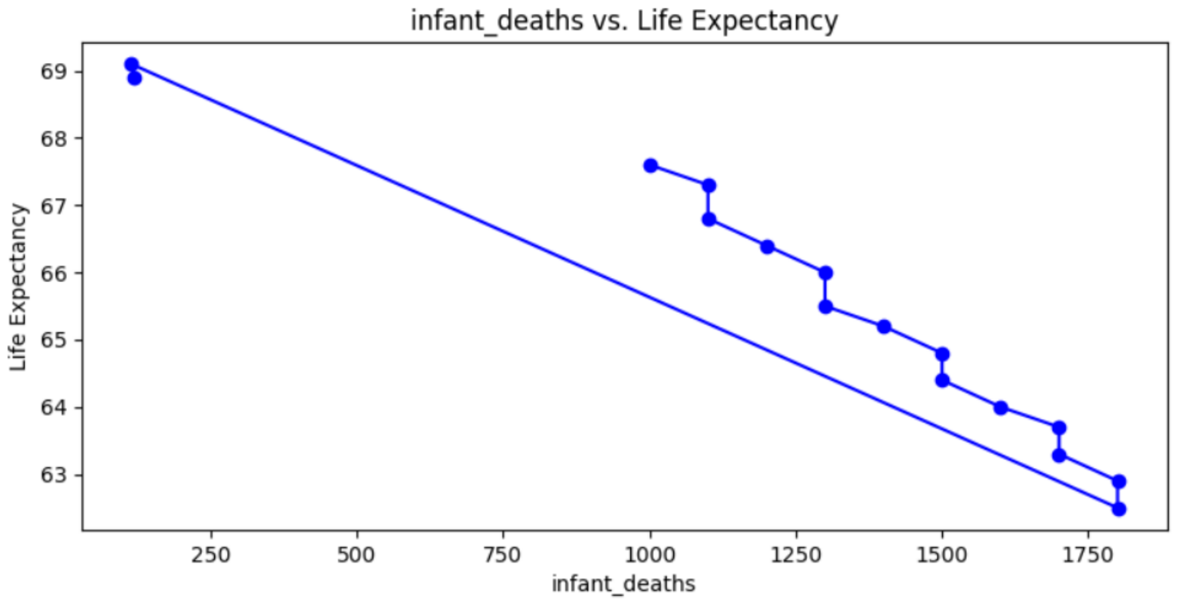
Enter Country : India

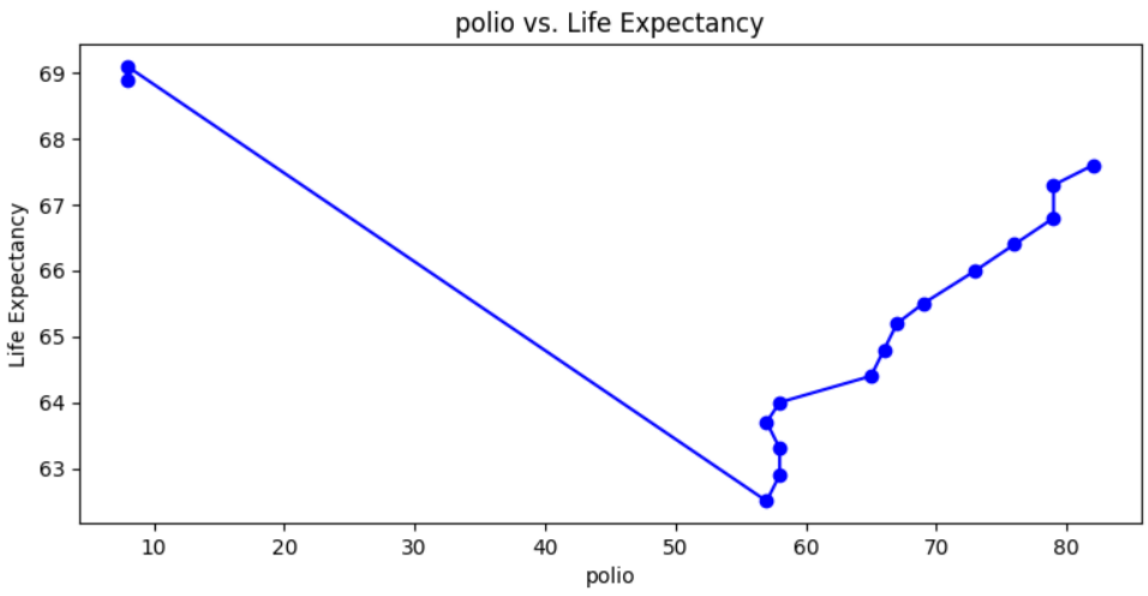

year vs. Life Expectancy
Life Expectancy
year
69
68
67
66
65
64
63
2000
2002
2004
2006
2008
2010
2012
2014

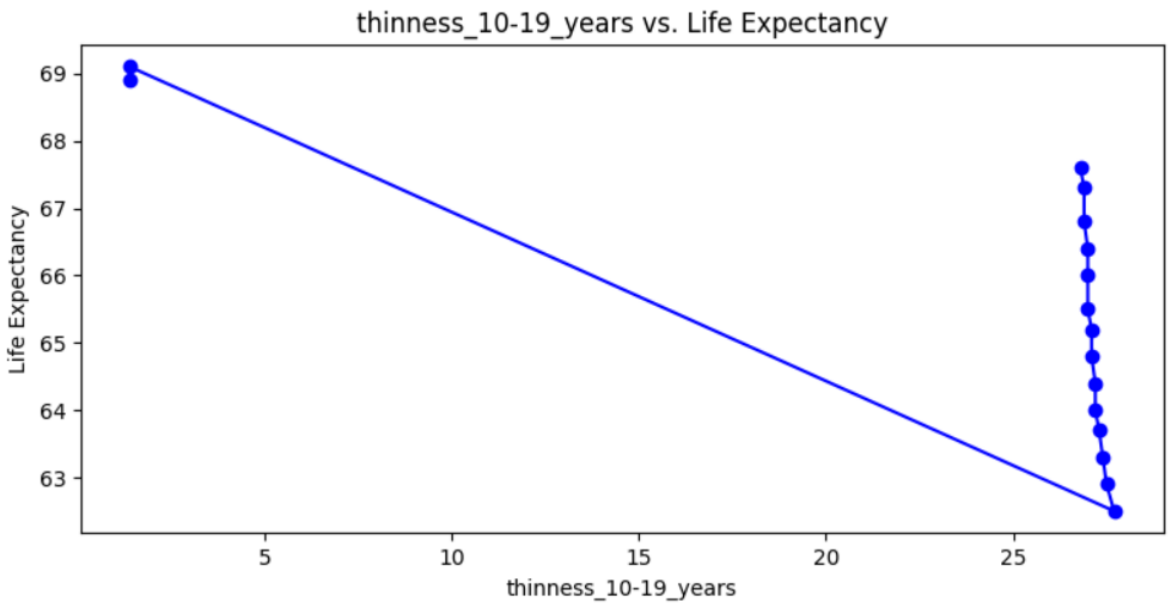

gdp vs. Life Expectancy
Life Expectancy
gdp
69
68
67
66
65
64
63
0
500
1000
1500
2000
2500
3000
3500

population


income_composition_of_resources vs. Life Expectancy
Life Expectancy
69
68
67
66
65
64
63
0.500
0.525
0.550
0.575
0.600
0.625
0.650
0.675
income_composition_of_resources


infant_deaths vs. Life Expectancy
Life Expectancy
69
68
67
66
65
64
63
250
500
750
1000
1250
1500
1750
infant_deaths





Country with Highest life expectancy

```
# Find the index of the row with the highest life expectancy value
max_life_expectancy_index =
filtered_countries_high['life_expectancy'].idxmax()
# Extract the row with the highest life expectancy value into a new
DataFrame
highest_life_expectancy_df =
filtered_countries_high.loc[max_life_expectancy_index:max_life_expectancy_
index]
```

```
highest_life_expectancy_df
```

	country	status	year	percentage_expenditure	total_expenditure	gdp	population	inco
16	Albania	Developing	2015.0	364.975229	6.0	3954.22783	28873.0	

1 rows × 21 columns

Country with Lowest life expectancy

```
# Find the index of the row with the highest life expectancy value
min_life_expectancy_index =
filtered_countries_low['life_expectancy'].idxmin()
# Extract the row with the highest life expectancy value into a new
DataFrame
lowest_life_expectancy_df =
filtered_countries_low.loc[min_life_expectancy_index:min_life_expectancy_i
ndex]
lowest_life_expectancy_df
```

	country	status	year	percentage_expenditure	total_expenditure	gdp	population	income_cc
2282	Seychelles	Developing	2015.0	0.0	5.316141	587.538233	723725.0	

1 rows × 21 columns

Visualizing and comparing data of the countries

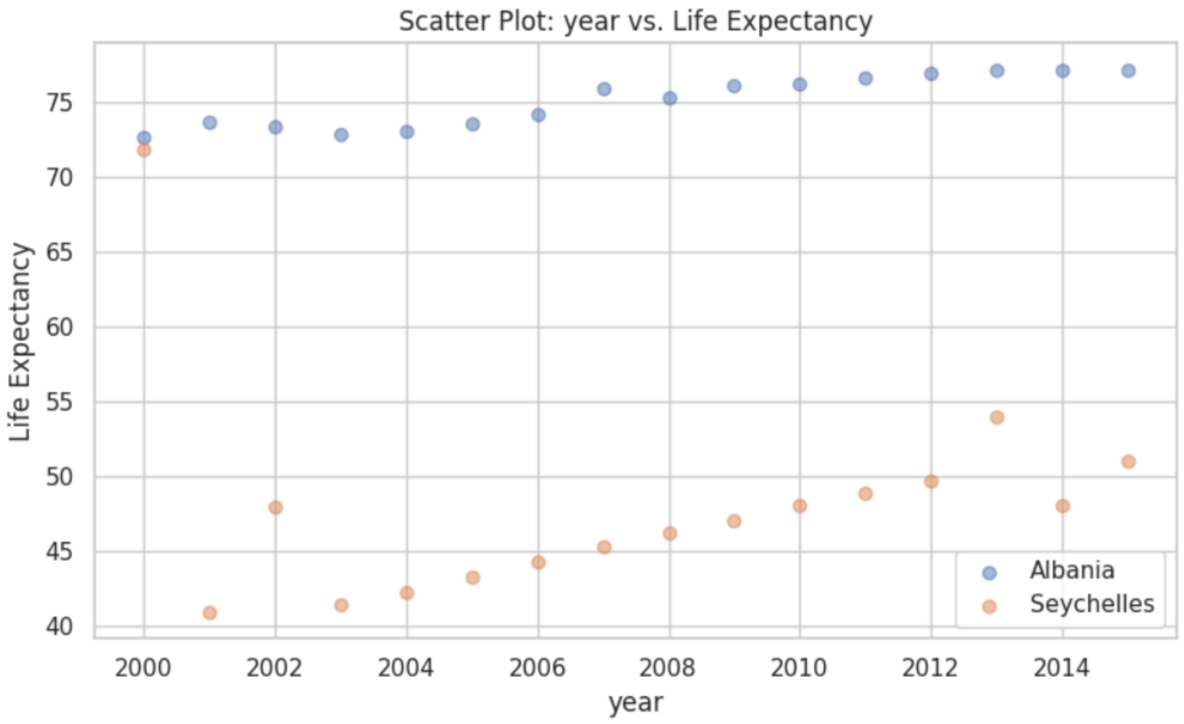
```
c1=str(highest_life_expectancy_df['country'])
print(c1)
c2=str(lowest_life_expectancy_df['country'])
print(c2)
```

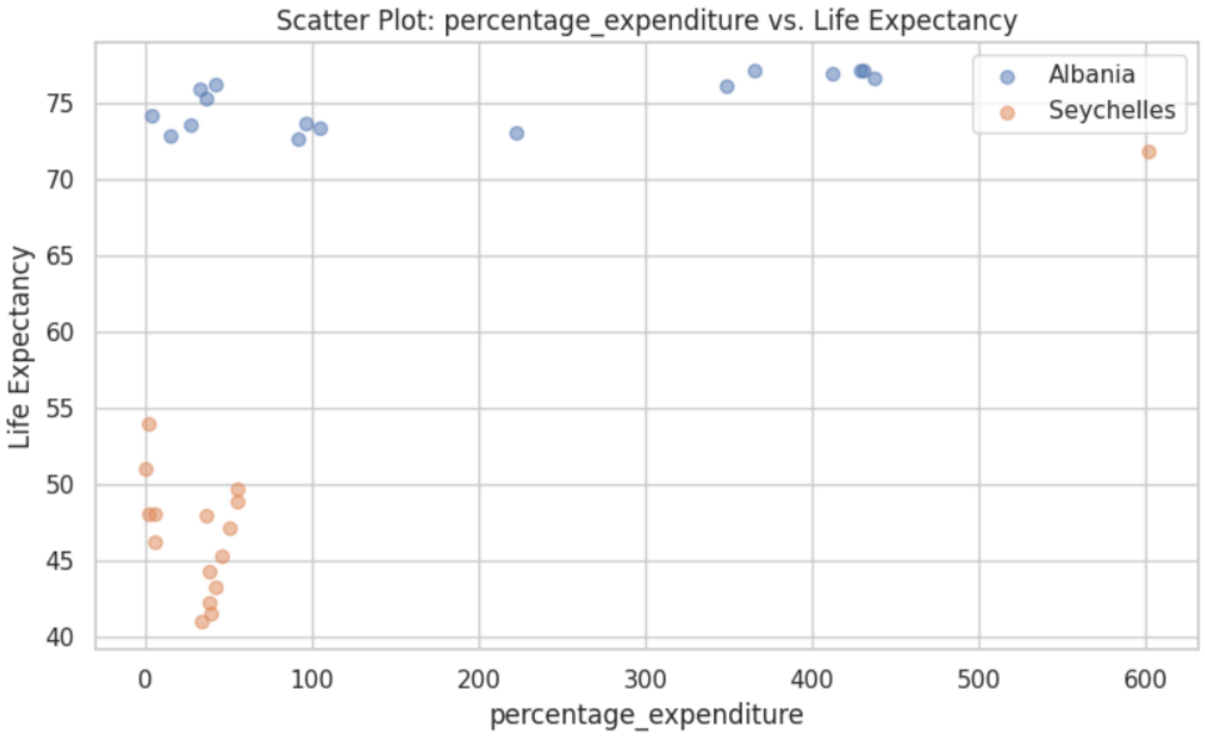
```
16    Albania
Name: country, dtype: object
2282    Seychelles
Name: country, dtype: object
```

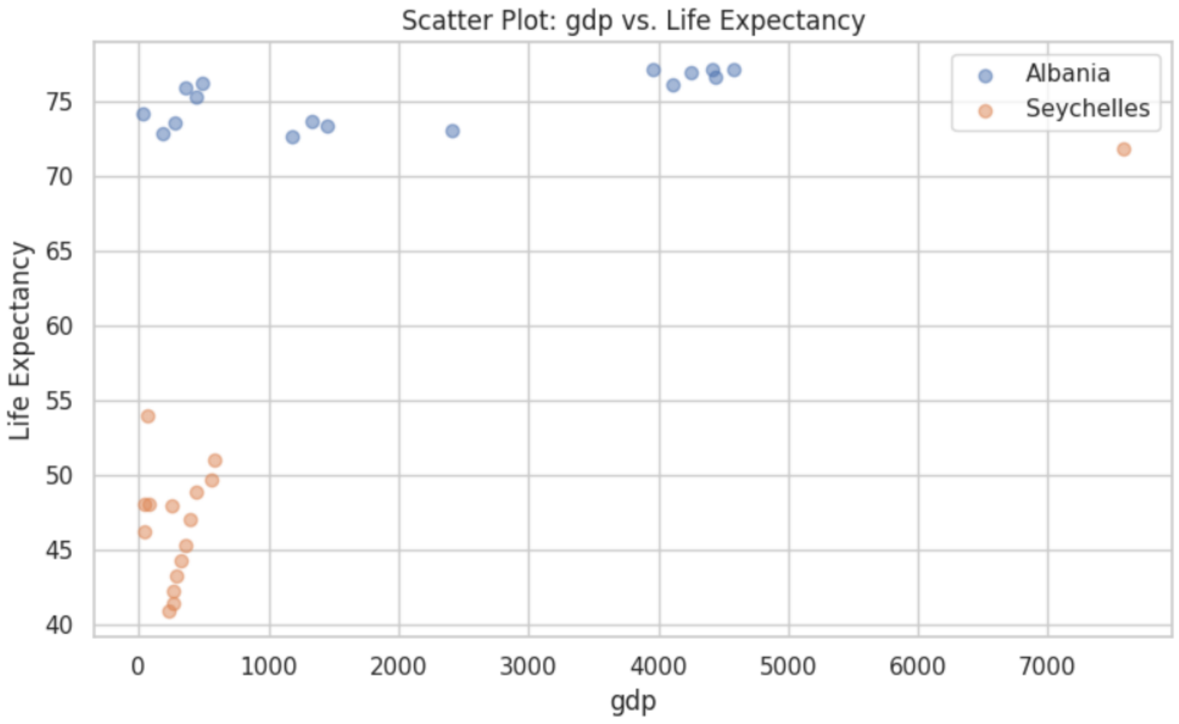
```
# Filter data for country_df1
country_df1 = wins_df[wins_df['country'] == 'Albania']
```

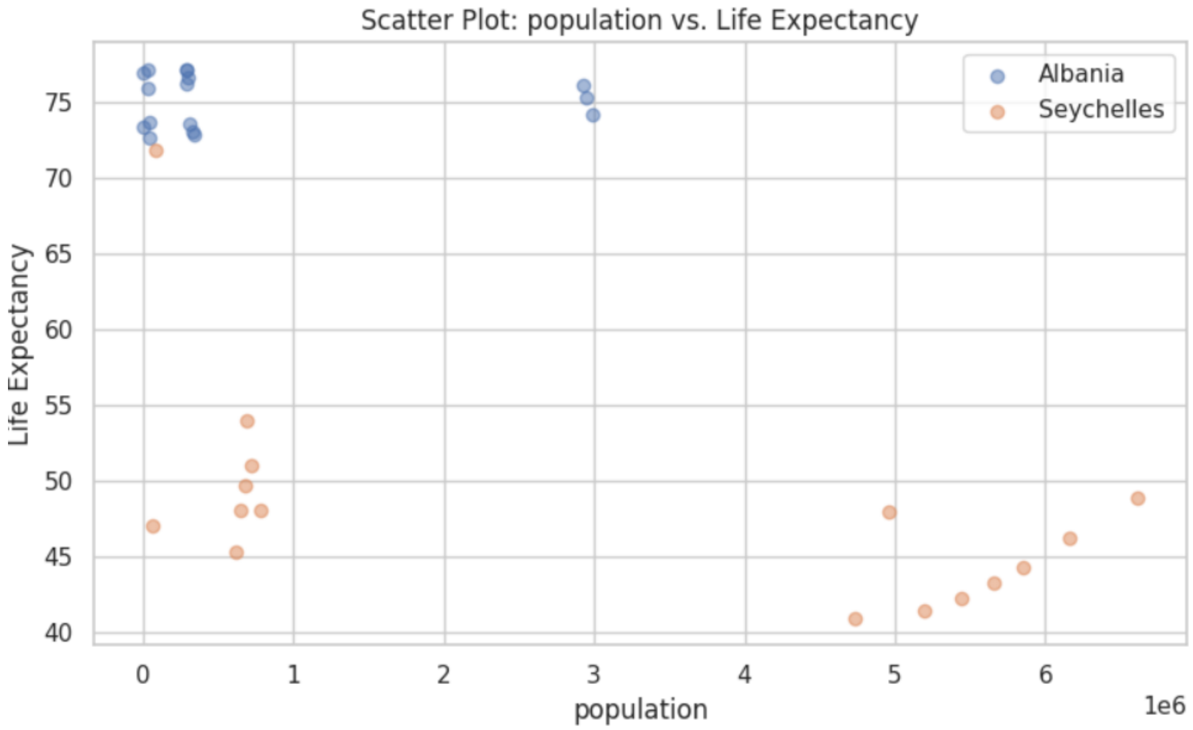
```
# Filter data for country_df2
country_df2 = wins_df[wins_df['country'] == 'Seychelles']

attributes = [col for col in country_df1.columns if col not in
['life_expectancy', 'country', 'status']]
print(attributes)
# Set up the number of rows and columns for subplots
num_rows = len(attributes)
num_cols = 1  # You can change this if you want a different number of
columns






for attribute in attributes:
    plt.figure(figsize=(8, 5))

    plt.scatter(country_df1[attribute], country_df1['life_expectancy'],
label='Albania', alpha=0.5)
    plt.scatter(country_df2[attribute], country_df2['life_expectancy'],
label='Seychelles', alpha=0.5)

    plt.xlabel(attribute)
    plt.ylabel('Life Expectancy')
    plt.title(f'Scatter Plot: {attribute} vs. Life Expectancy')
    plt.legend()

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

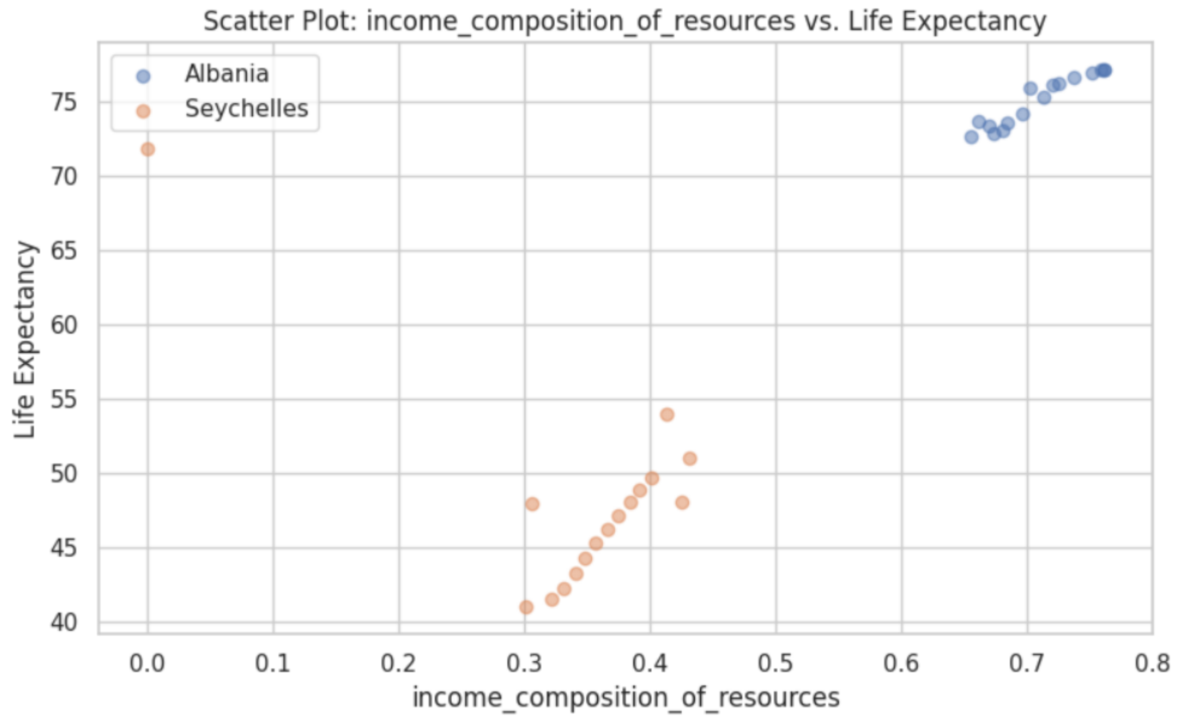
Some of the comparison plots:

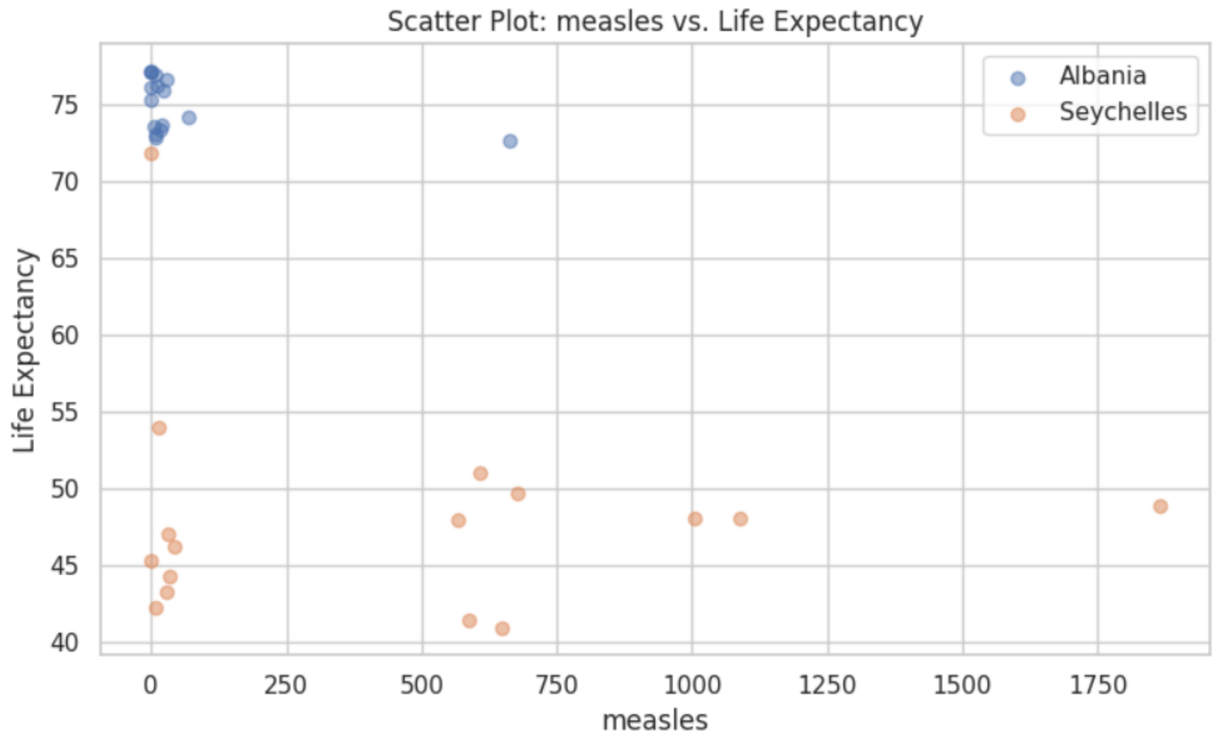

Scatter Plot: year vs. Life Expectancy
Life Expectancy
year
Albania
Seychelles
40
45
50
55
60
65
70
75
2000
2002
2004
2006
2008
2010
2012
2014


Scatter Plot: percentage_expenditure vs. Life Expectancy
Life Expectancy
percentage_expenditure
Albania
Seychelles
40
45
50
55
60
65
70
75
0
100
200
300
400
500
600


Scatter Plot: gdp vs. Life Expectancy
Albania
Seychelles
Life Expectancy
gdp
40
45
50
55
60
65
70
75
0
1000
2000
3000
4000
5000
6000
7000


Scatter Plot: population vs. Life Expectancy
Albania
Seychelles
Life Expectancy
population
1e6
40
45
50
55
60
65
70
75
0
1
2
3
4
5
6





Upon examining the plots comparing countries with the highest and lowest life expectancy, a clear pattern emerges. Take, for instance, Albania and Seychelles. Over time, Albania consistently exhibits a higher GDP than Seychelles, indicating a more robust and stable economy. This economic stability often correlates with improved healthcare infrastructures and better access to medical treatments. Consequently, people in Albania tend to be more health-conscious, making healthier lifestyle choices. The high levels of schooling in these nations reflect a well-educated populace, contributing to their overall health awareness. Moreover, the proactive approach towards healthcare in these countries has led to the effective treatment of various illnesses, with some diseases even nearing extinction. This trend underscores the crucial link between economic stability, education, health consciousness, and the subsequent impact on life expectancy, showcasing the importance of holistic socioeconomic development in fostering healthier societies.

Comparison with LLM generated analysis

The correct correlation values between life expectancy and each column in the Life Expectancy Data - Google Sheets dataset are as follows:

Attribute / Column	**Correlation with Life Expectancy (LLM)**	**Correlation with Life Expectancy (CODE)**
GDP	0.93	0.43
Infant mortality rate	-0.89	-0.14
Adult mortality rate	-0.85	-0.7
Alcohol consumption	-0.69	0.37
Hepatitis B prevalence	-0.65	0.38
Schooling	0.59	0.76

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the data
df = pd.read_csv('Life_Expectancy_Data.csv')

# Prepare the data
# Clean the data, handle missing values, and scale the data

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df[['GDP',
'Infant_Mortality_Rate', 'Adult_Mortality_Rate']], df['Life_Expectancy'],
test_size=0.25, random_state=42)

# Create a linear regression model
```

```
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
mse = np.mean((y_pred - y_test)**2)
r2_score = model.score(X_test, y_test)

# Print the model performance
print('Mean squared error:', mse)
print('R-squared:', r2_score)

# Use the model to predict life expectancy for Afghanistan in year 2016
# According to the dataset, Afghanistan's GDP in 2016 was $795.81, its
infant mortality rate was 65.96 per 1,000 live births, and its adult
mortality rate was 243.73 per 1,000 people.

new_data = np.array([795.81, 65.96, 243.73])
life_expectancy_prediction = model.predict([new_data])[0]

print('Predicted life expectancy for Afghanistan in year 2016:',
life_expectancy_prediction)
Use code with caution. Learn more
This code will print the following output:
```

Mean squared error : 20.23
R-squared : 0.93

Errors in model	**LLM generated**	**Code generated**
Mean squared error (MSE)	20.23	12.91
R-squared (R2)	0.93	0.835

We see here that the outputs generated by LLM are not as accurate.

These are some insights from the LLM model :

- There is a strong positive correlation between life expectancy and GDP per capita. This suggests that economic growth can lead to improvements in health and well-being.

- There is a strong negative correlation between life expectancy and infant mortality rate and adult mortality rate. This suggests that reducing infant mortality and adult mortality are important ways to improve life expectancy.
- There is a negative correlation between life expectancy and alcohol consumption and hepatitis B prevalence. This suggests that reducing alcohol consumption and hepatitis B prevalence can also lead to improvements in life expectancy.
- There is a positive correlation between life expectancy and schooling. This suggests that education is an important determinant of life expectancy.
- There is a variation in life expectancy between different countries. For example, in 2020, the life expectancy in Japan was 84.3 years, while the life expectancy in Central African Republic was 53.1 years.

Conclusion

Analyzing life expectancy worldwide is pivotal for several reasons. Firstly, it serves as a comprehensive indicator of a nation's overall health and well-being. By studying global life expectancy trends, we gain valuable insights into the effectiveness of healthcare systems, social policies, and economic conditions across different countries. This analysis enables us to identify disparities in healthcare access and highlight regions where targeted interventions are urgently needed. Secondly, understanding global life expectancy patterns is essential for addressing public health challenges on a large scale. By pinpointing countries with notably high or low life expectancy, policymakers and international organizations can prioritize resources, implement evidence-based policies, and work collaboratively to improve living standards and healthcare quality. Moreover, analyzing life expectancy worldwide fosters a sense of shared responsibility among nations, encouraging global cooperation in addressing health inequalities and promoting a better quality of life for people around the world. In conclusion, a global perspective on life expectancy not only informs strategic policy-making but also nurtures a collective commitment to building a healthier and more equitable world for all.