Life Expectancy Analysis

3 steps to this analysis:

1. Data Cleaning
2. Data Exploration - Visualisation
3. Summary

Loading in the datasets

```
import pandas as pd
import numpy as np
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

#Load data and manipulate as needed
df = pd.read_csv("/content/led.csv")

df.head()
```

	Country	Year	Status	Lifeexpectancy	AdultMortality	infantdeaths	Alcohol
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01

5 rows × 22 columns

Section 1. Data Cleaning

It is important to understand the variables presented in this data. To gain a deeper understanding there are 3 very important questions:

1. What is the meaning of the variable and what type of variable is it, i.e., nominal, ordinal, interval, or ratio?
2. Are there any missing values in the variable? If yes, how should they be handled?
3. Are there any outliers in the variable? If yes, how should they be addressed?

1.1: Dataset Description/Variable Descriptions

Below is a list of the variable descriptions from the World Health Organisation (WHO)

I have also cleaned up the variable names.

```
orig_cols = list(df.columns)
new_cols = []
for col in orig_cols:
    new_cols.append(col.strip().replace('  ', ' ').replace(' ', '_').lower())
df.columns = new_cols
df.columns
```

```
Index(['country', 'year', 'status', 'lifeexpectancy', 'adultmortality',
       'infantdeaths', 'alcohol', 'percentageexpenditure', 'hepatitisb',
       'measles', 'bmi', 'under-fivedeaths', 'polio', 'totalexpenditure',
       'diphtheria', 'hiv/aids', 'gdp', 'population', 'thinness1-19years',
       'thinness5-9years', 'incomecompositionofresources', 'schooling'],
      dtype='object')
```

Variable Descriptions

- 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
- lifeexpectancy (Ratio) - the life expectancy of people in years for a particular country and year
- adultmortality (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%
- infantdeaths (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)
- hepatitisb (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-fivedeaths (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
- totalexpenditure (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
- thinness1-19_years (Ratio) - rate of thinness among people aged 10-19 (Variable will be renamed to thinness10-19_years to more accurately represent the variable)
- thinness5-9_years (Ratio) - rate of thinness among people aged 5-9
- incomecompositionofresources (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

```
#Change the name of the variable thinness_1-19_years to thinness_10-19_years - more accurate
df.rename(columns={'thinness1-19years':'thinness10-19years'}, inplace=True)
```

1.2: Missing Values

There are few things that must be done concerning missing values:

1. Detection of missing values
    - Find nulls
    - Could a null be signified by anything other than null? Zero values perhaps?
2. Dealing with missing values
    - Fill nulls? Impute or Interpolate
    - Eliminate nulls?

1. Missing Values Detection

- What values could be null?
- What values could be erroneous?

Going to use df.describe() to look at each variable on its own to see if the values make sense given the description of the variable.

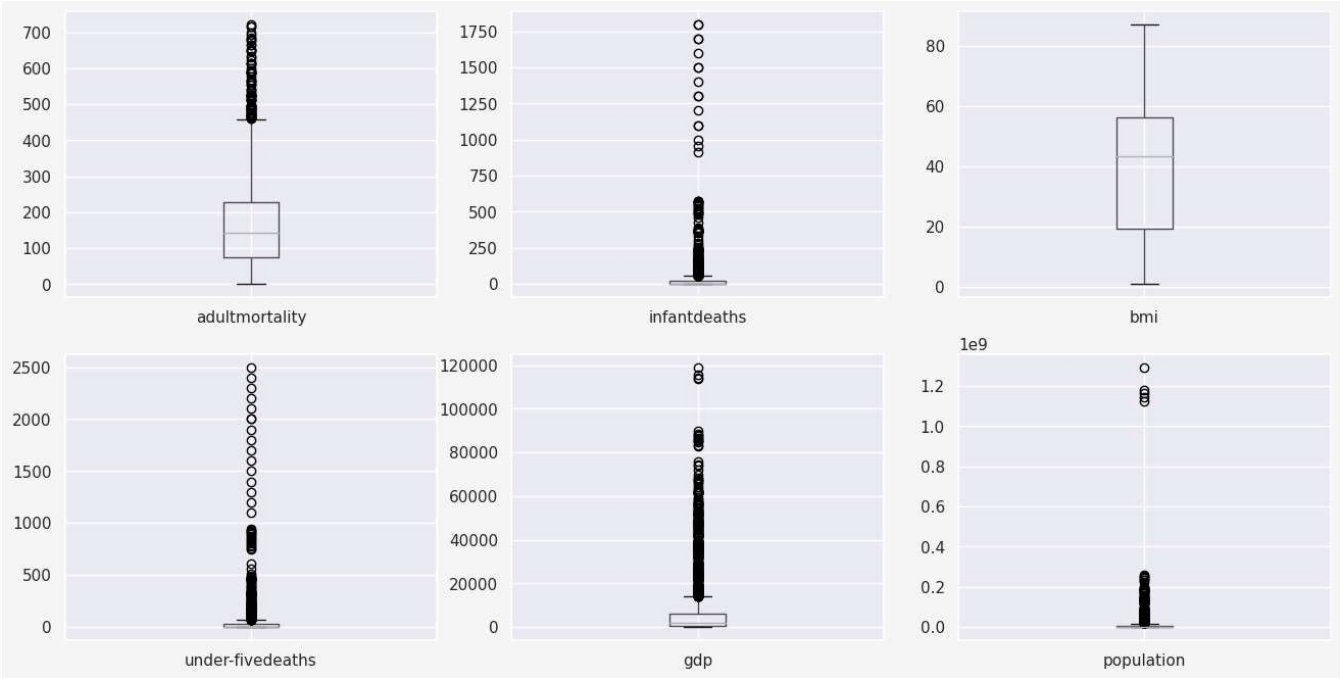
```
df.describe().iloc[:, 1:]
```

	lifeexpectancy	adultmortality	infantdeaths	alcohol	percentageexpenditure	hepatitisb	measles	bm
count	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.00000
mean	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.32124
std	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.04403
min	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.00000
25%	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.30000

Things that may not make sense from above:

- Minimum Adult mortality of 1 - this is likely an error in measurement, but what values make sense here? May need to change to null if under a certain threshold.
- Minimum Infant deaths as low as 0 per 1000 - not possible. Going to change these values to null. Also on the other end 1800 is likely an outlier, but it is possible in a country with very high birthrates and perhaps a not very high population total - this can be dealt with later.
- Lowest BMI of 1 and highest 87.3? A BMI of 15 or lower is dangerously underweight and a BMI of 40 or higher is morbidly obese, therefore a large number of these measurements seems unrealistic. This variable might not be very good quality
- Minimum Under Five Deaths as low as 0 - similar to infant deaths, not possible
- GDP per capita as low as 1.68 (USD) - Doubtful - but perhaps values this low are outliers.
- Population of 34 for an entire country seems off

```
plt.figure(figsize=(16,8))
for i, col in enumerate(['adultmortality', 'infantdeaths', 'bmi', 'under-fivedeaths', 'gdp', 'population'], start=1):
    plt.subplot(2, 3, i)
    df.boxplot(col)
```



Some of the values above may be considered outliers, but there are some that are almost certainly errors. For the following variables, null values will be assigned because the values do not appear to be valid:

- Adult mortality rates lower than the 5th percentile
- Infant deaths of 0
- BMI less than 10 and greater than 50
- Under Five deaths of 0

```
#calculate the 5th percentile of non-null values
mort_5_percentile = np.percentile(df.adultmortality.dropna(), 5)

#Replaces any adultmortality value less than the 5th percentile value with NaN (missing value)
df.adultmortality = df.apply(lambda x: np.nan if x.adultmortality < mort_5_percentile else x.adultmortality, axis=1)
```

```
#Replace all 0 values with NaN
df.infantdeaths = df.infantdeaths.replace(0, np.nan)

#Replaces any values less than 10 or higher than 50 with NaN
df.bmi = df.apply(lambda x: np.nan if (x.bmi < 10 or x.bmi > 50) else x.bmi, axis=1)

#Replace all 0 values with NaN
df['under-fivedeaths'] = df['under-fivedeaths'].replace(0, np.nan)
```

Check for nulls:

```
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,

nulls_breakdown()
```

```
 [iloc = 3] lifeexpectancy has 10 null values: 0.34% null
 [iloc = 4] adultmortality has 155 null values: 5.28% null
 [iloc = 5] infantdeaths has 848 null values: 28.86% null
 [iloc = 6] alcohol has 194 null values: 6.6% null
 [iloc = 8] hepatitisb has 553 null values: 18.82% null
 [iloc = 10] bmi has 1456 null values: 49.56% null
 [iloc = 11] under-fivedeaths has 785 null values: 26.72% null
 [iloc = 12] polio has 19 null values: 0.65% null
 [iloc = 13] totalexpenditure has 226 null values: 7.69% null
 [iloc = 14] diphtheria has 19 null values: 0.65% null
 [iloc = 16] gdp has 448 null values: 15.25% null
 [iloc = 17] population has 652 null values: 22.19% null
 [iloc = 18] thinness10-19years has 34 null values: 1.16% null
 [iloc = 19] thinness5-9years has 34 null values: 1.16% null
 [iloc = 20] incomecompositionofresources has 167 null values: 5.68% null
 [iloc = 21] schooling has 163 null values: 5.55% null
 Out of 22 total columns, 16 contain null values; 72.73% columns contain null values.
```

2. Dealing with Missing Values

Nearly half of the BMI variable's values are null therefore it is best to remove this variable

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

There are numerous columns with missing values in the time series data. Since the data is sorted by country, interpolating the missing values by country may seem like a reasonable approach. However, since the data for all the null values is null for each year within a country, interpolation by country does not fill in any values. Thus, imputing the missing values with the mean value for each year may be the most appropriate method. The code below performs mean imputation for each year.

```
#Set up blank list
imputed_data = []
for year in list(df.year.unique()):
    #created year_data that contains only the records from df that match the current year in the loop
    year_data = df[df.year == year].copy()
    for col in list(year_data.columns)[3:]:
        #Replaces missing values with the mean value for that year
        year_data[col] = year_data[col].fillna(year_data[col].dropna().mean()).copy()
    imputed_data.append(year_data)
#Concat to stacking the dataframes vertically - on top of each other
df = pd.concat(imputed_data).copy()

#Check for nulls again
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,

    nulls_breakdown()
```

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

1.3 Outliers

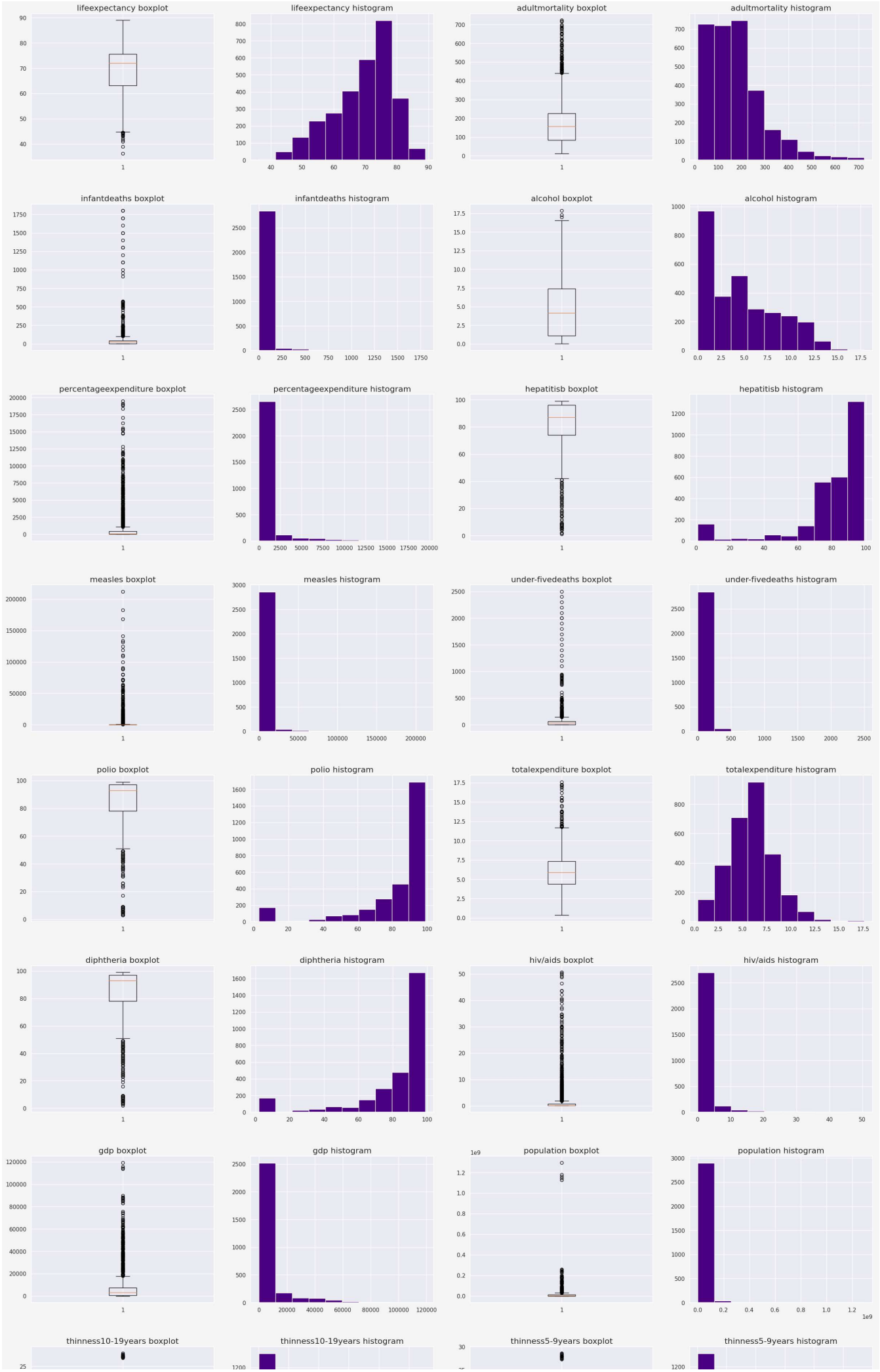
Similar to missing values, there are a few things that need done in order to deal with outliers:

1. Detect the outliers
    - Boxplots/histograms
    - Tukey's Method
2. Deal with outliers
    - Drop outliers?
    - Limit/Winsorise outliers?
    - Transform the data using log/inverse/square root/etc?

1. Outliers Detection

First a boxplot and histogram will be created for each continuous variable in order to visually see if outliers exist.

```
# Select continuous variables from the dataframe
cont_vars = list(df.columns)[3:]

def visualize_outliers(data):
    # Define the size of the figure and the number of subplots per row and column
    fig = plt.figure(figsize=(30, 60))
    rows, cols = 9, 4

    i = 0
    for col in cont_vars:
        i += 1
        plt.subplot(9, 4, i)
        plt.boxplot(data[col])
        plt.title('{} boxplot'.format(col), fontsize = 16)
        i += 1
        plt.subplot(9, 4, i)
        plt.hist(data[col], color='indigo')
        plt.title('{} histogram'.format(col), fontsize = 16)

        # Adjust the spacing between subplots and display the figure
        plt.subplots_adjust(hspace=0.3)

    plt.show()

visualize_outliers(df)
```


lifeexpectancy boxplot
lifeexpectancy histogram
adultmortality boxplot
adultmortality histogram
infantdeaths boxplot
infantdeaths histogram
alcohol boxplot
alcohol histogram
percentageexpenditure boxplot
percentageexpenditure histogram
hepatitisb boxplot
hepatitisb histogram
measles boxplot
measles histogram
under-fivedeaths boxplot
under-fivedeaths histogram
polio boxplot
polio histogram
totalexpenditure boxplot
totalexpenditure histogram
diphtheria boxplot
diphtheria histogram
hiv/aids boxplot
hiv/aids histogram
gdp boxplot
gdp histogram
population boxplot
population histogram
thinness10-19years boxplot
thinness10-19years histogram
thinness5-9years boxplot
thinness5-9years histogram

By visual inspection, it is evident that there exist several outliers for all the variables, including the target variable of life expectancy. The application of Tukey's method below will confirm this statistically, where anything beyond 1.5 times the Interquartile Range (IQR) is considered an outlier.

```
#find outliers by specified criteria
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)
```

```
---------------lifeexpectancy---------------
Number of outliers: 17
Percent of data that is outlier: 0.58%
---------------adultmortality---------------
Number of outliers: 97
Percent of data that is outlier: 3.3%
---------------infantdeaths---------------
Number of outliers: 135
Percent of data that is outlier: 4.59%
---------------alcohol---------------
Number of outliers: 3
Percent of data that is outlier: 0.1%
---------------percentageexpenditure---------------
Number of outliers: 389
Percent of data that is outlier: 13.24%
---------------hepatitisb---------------
Number of outliers: 222
Percent of data that is outlier: 7.56%
---------------measles---------------
Number of outliers: 542
Percent of data that is outlier: 18.45%
---------------under-fivedeaths---------------
Number of outliers: 142
Percent of data that is outlier: 4.83%
---------------polio---------------
Number of outliers: 279
Percent of data that is outlier: 9.5%
---------------totalexpenditure---------------
Number of outliers: 51
Percent of data that is outlier: 1.74%
---------------diphtheria---------------
Number of outliers: 298
Percent of data that is outlier: 10.14%
---------------hiv/aids---------------
Number of outliers: 542
Percent of data that is outlier: 18.45%
---------------gdp---------------
Number of outliers: 300
Percent of data that is outlier: 10.21%
---------------population---------------
Number of outliers: 203
Percent of data that is outlier: 6.91%
---------------thinness10-19years---------------
Number of outliers: 100
Percent of data that is outlier: 3.4%
---------------thinness5-9years---------------
Number of outliers: 99
Percent of data that is outlier: 3.37%
---------------incomecompositionofresources---------------
Number of outliers: 130
Percent of data that is outlier: 4.42%
---------------schooling---------------
Number of outliers: 77
Percent of data that is outlier: 2.62%
```

2. Dealing with Outliers

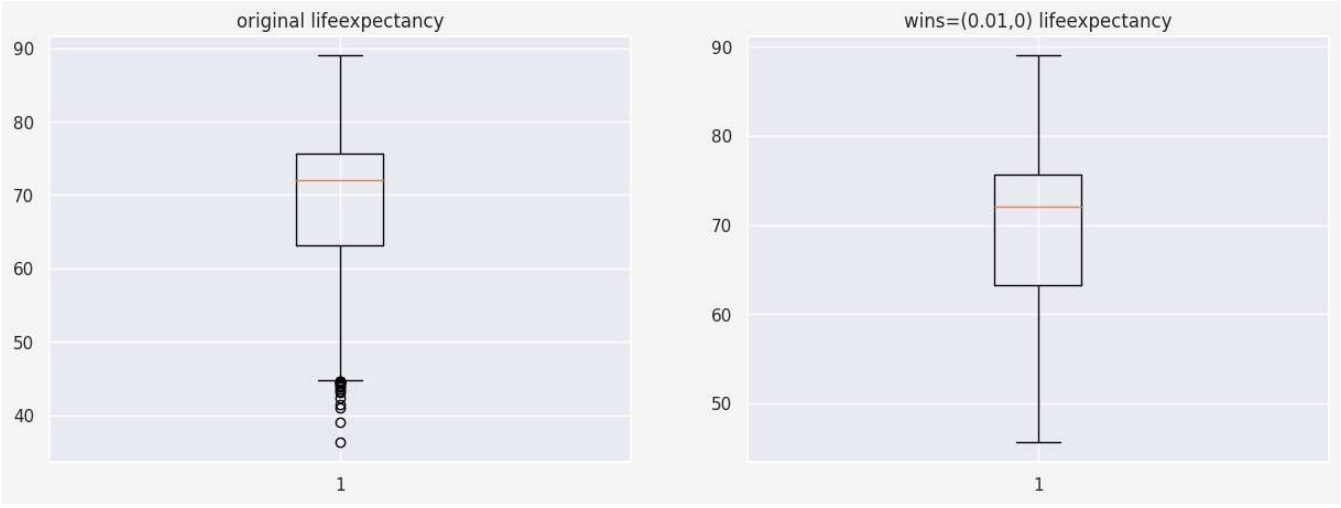
There are a number of ways to deal with outliers in a dataset, the usual options are as follows:

- Drop Outliers (best avoided in order to keep as much information as possible)
- Limit values to upper and/or lower bounds (Winsorize the data)

- Transform the data (log/inverse/square root/etc.) advantage: can 'normalize' the data and eliminate outliers disadvantage: cannot be done to variables containing values of 0 or below

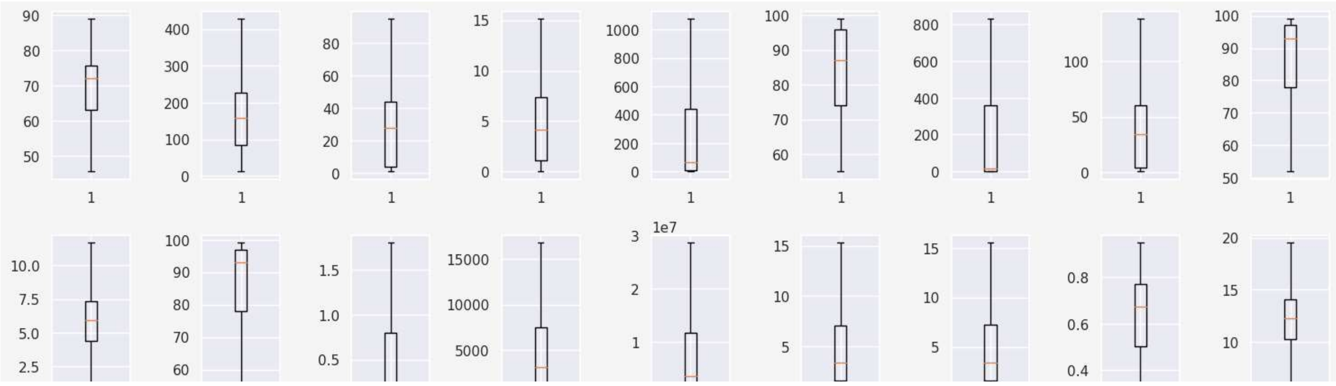
As each variable has a distinct number of outliers and these outliers are situated on different sides of the data, it is advisable to use winsorisation to restrict the values for each variable until all outliers are eliminated. To accomplish this, the function below processes each variable individually and permits the use of either a lower or upper limit for winsorisation. The function displays two boxplots (one displaying the original data and the other displaying the winsorised data) side by side for each variable by default. Once an acceptable limit is identified through visual examination, the winsorised data is saved in the wins_dict dictionary, allowing easy access to the data later on.

```
#Testing different upper/lower limits to find acceptable limits
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)
```



All the variables have now been winsorised as little as possible in order to keep as much data in tact as possible while still being able to eliminate the outliers. Finally, small boxplots will be shown for each variable's winsorized data to show that the outliers have indeed been dealt with.

Show code



Creating the new dataframe with the winsorised data

```
df_win = df.iloc[:, 0:3]
for col in cont_vars:
    df_win[col] = wins_dict[col]
```

Section 2: Data Exploration - Visualisation

Descriptive Statistics

```
df_win.describe()
```

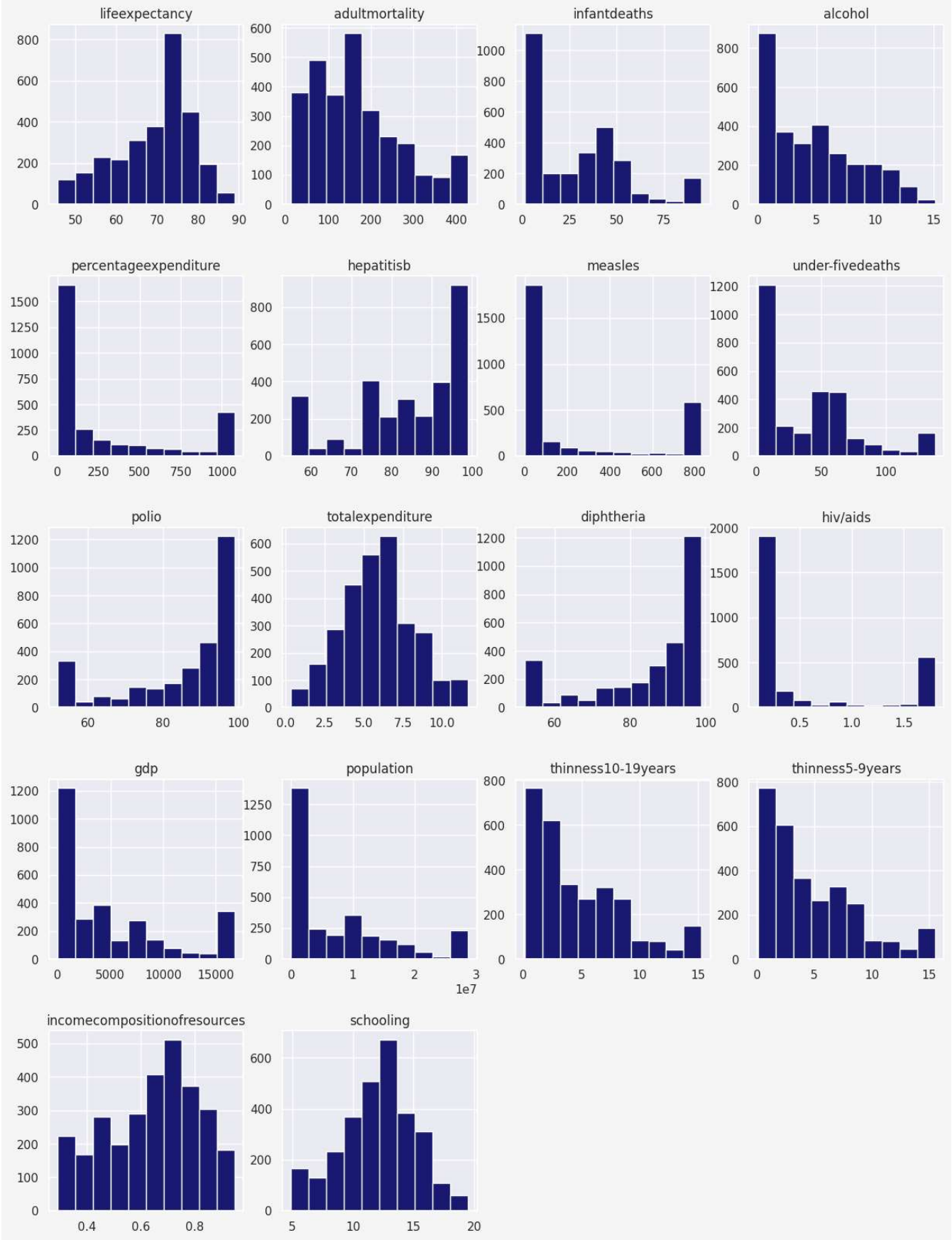
```
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:4650: UserWarning: Warning: 'partition' will ignore the 'ma
```

	year	lifeexpectancy	adultmortality	infantdeaths	alcohol	percentageexpenditure	hepatitisb	measles
count	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000
mean	2007.518720	69.248795	169.191269	28.607323	4.640667	281.501093	83.671293	220.691287
std	4.613841	9.462136	107.646049	25.699534	3.910036	384.161450	13.848933	329.192321
min	2000.000000	45.600000	13.000000	1.000000	0.010000	0.000000	55.000000	0.000000
25%	2004.000000	63.200000	84.000000	4.000000	1.092500	4.685343	74.099237	0.000000
50%	2008.000000	72.000000	157.000000	28.000000	4.140000	64.912906	87.000000	17.000000
75%	2012.000000	75.600000	227.000000	43.766917	7.390000	441.534144	96.000000	360.250000
max	2015.000000	89.000000	428.000000	95.000000	15.140000	1077.712092	99.000000	831.000000

Visual Distributions

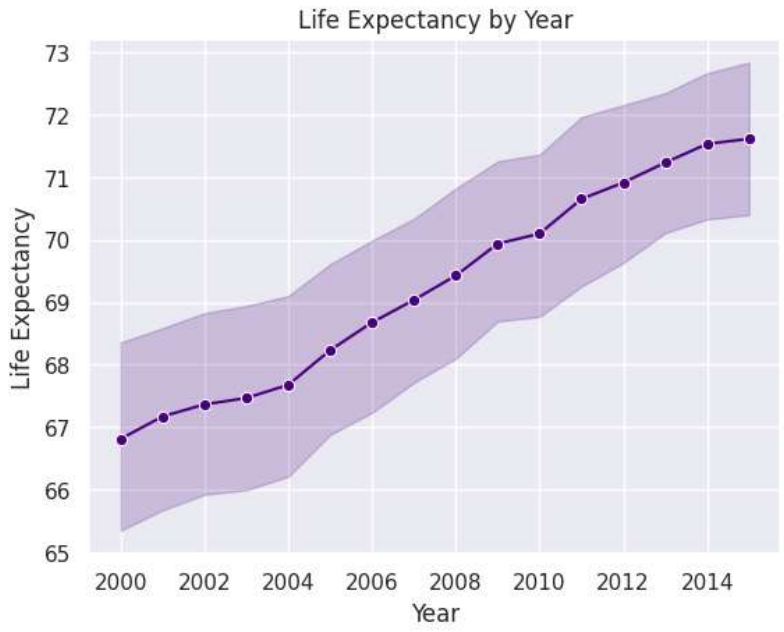
```
# Creating the histograms
plt.figure(figsize=(15, 20))
# Iterate over the columns and plot a histogram for each
for i, col in enumerate(cont_vars, 1):
    # Set the number of rows and columns in the plot grid and selects the current plot to be i
    plt.subplot(5, 4, i)
    plt.hist(df_win[col], color = 'midnightblue')
```

```
    plt.title(col)
    # Adjust the spacing between the subplots
    plt.subplots_adjust(hspace=0.4)
```



Life Expectancy over time

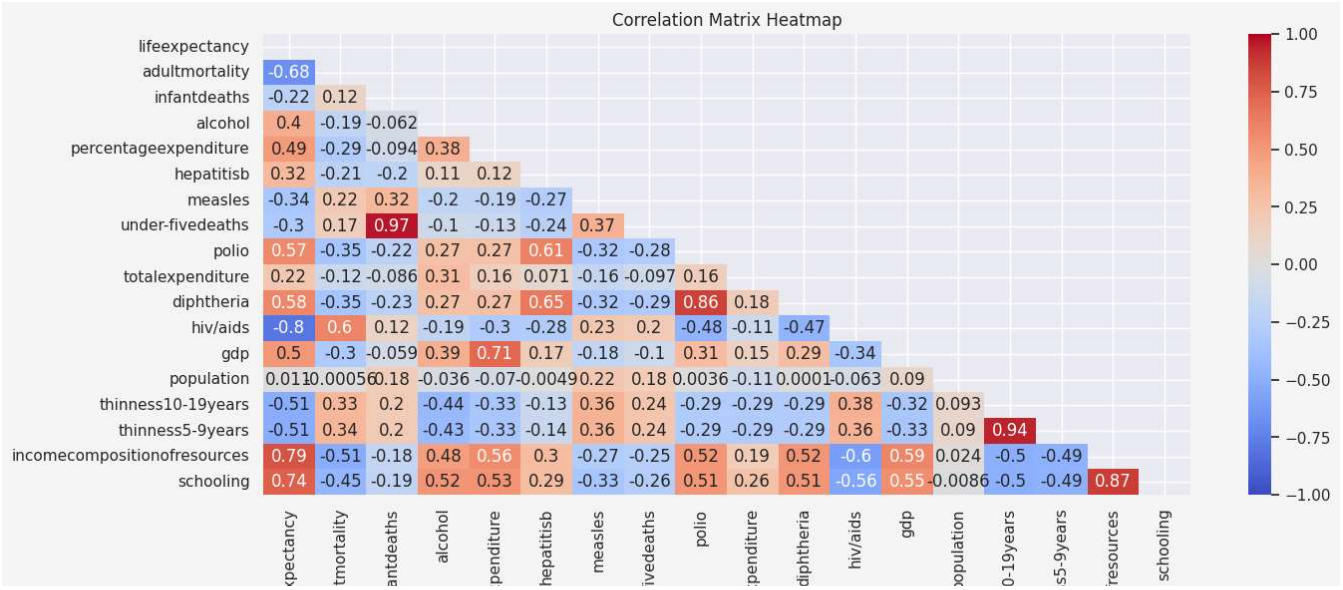
```
# Set the Seaborn theme
sns.set_theme()

# Create a scatter plot
sns.lineplot(x="year", y="lifeexpectancy", data=df_win, color='indigo', marker = 'o')

# Add a title and axis labels to the plot
plt.title('Life Expectancy by Year')
plt.xlabel('Year')
plt.ylabel('Life Expectancy')

# Set the plot background color to pale grey
plt.rcParams["figure.facecolor"] = "#F5F5F5"

# Set the gridline color to white
plt.rcParams["grid.color"] = "white"

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



There appears to be a positive trend over time with life expectancy increasing gradually year on year.

Continuous Variable Correlation

```
#Create a heatmap of the correlation matrix for the continuous variables in the dataset.
def plot_correlation_matrix(data):
    # Compute the correlation matrix and create a mask to show only the upper triangle
    corr = df_win[cont_vars].corr()
    mask = np.triu(np.ones_like(corr, dtype=bool))

    # Define the size and style of the figure and plot the heatmap
    plt.figure(figsize=(15,6))
    sns.heatmap(corr, annot=True, fmt='.2g', vmin=-1, vmax=1, center=0, cmap='coolwarm', mask=mask)

    # Set the y-axis limits and add a title to the plot
    plt.ylim(len(cont_vars), 0)
    plt.title('Correlation Matrix Heatmap')
    plt.show()

# Call the function to plot the correlation matrix for the dataframe
plot_correlation_matrix(df_win)
```



We can see from the above heatmap a number of important correlations:

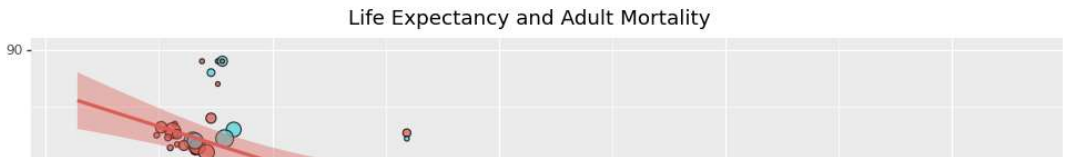
- Life Expectancy (target variable) appears to be relatively highly correlated with:
  - Adult Mortality (negative)
  - HIV/AIDS (negative)
  - Income Composition of Resources (positive)
  - Schooling (positive)
- The correlation between life expectancy and adult mortality with population is very weak
- The correlation between population and all variables is very weak except for with measles, infant deaths and under 5 deaths (still pretty low correlation with these 3 also but not as low)
- Infant deaths and under 5 deaths are very hightly correlated
- Polio vaccine rate is very highly corrected with diptheria vaccine rate
- Hepatitis B vaccione rate is highly correlated with both polio and diptheria vaccine rates
- Income composition of resources and schooling are very highly correlated
- Thinness 5-9 years is very highly correlated with thinness 10-19 years

Correlation Scatterplots

1. Life Expectancy and Adult Mortality

The chart below shows the correlation between life expectancy and adult mortality as a scatter plot. The year displayed is 2014.

Show code



Findings from the chart:

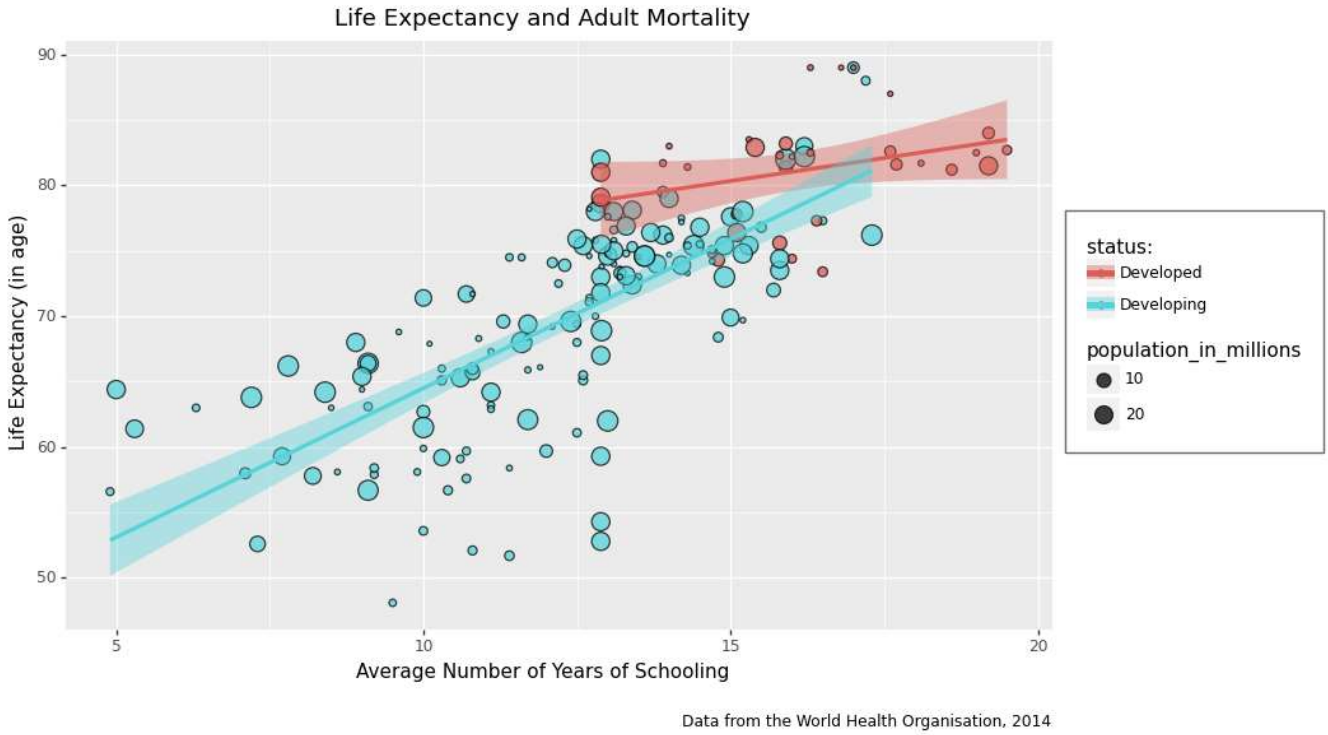
We can see clearly in this chart the correlation between life expectancy and adult mortality (number of people dying between 15 and 60 years old per 1000 population) that was given in the correlation matrix above.

Developed countries generally have longer life expectancies, whereas developing countries tend to have a higher probability of dying between the ages of 15 and 60. In both developing and developed countries, an increase in the probability of death between 15 and 60 years of age is associated with a decrease in life expectancy.

1. Life Expectancy and Schooling

The chart below shows the correlation between life expectancy and schooling as a scatter plot. The year displayed is 2014.

Show code




```
<ggplot: (8747015584799)>
```

We examine the influence of education on life expectancy. The evidence shows a clear positive correlation between the number of years of education and the average life expectancy, particularly in developing countries. Furthermore, developed countries exhibit notably higher levels of both education and life expectancy in comparison to developing countries.

1. Life Expectancy and GDP

The chart below shows the correlation between life expectancy and GDP as a scatter plot. The year displayed is 2014.

Show code