Life Expectancy Case Study

Business Context

The EU Allianz is an insurance company that is a high revenue-generating company, and the owner of this company is very serious about mitigating the risk factors involved in their business. The single most important factor used by insurance firms to determine life insurance premiums is life expectancy. The age at which a person is expected to die is known as life expectancy. There are several factors that affect life expectancy. Many studies have been undertaken in the past on factors affecting life expectancy. The company has prepared data to anticipate life expectancy in order to overcome this challenge. If a company can accurately anticipate life expectancy, it can save billions of dollars that would otherwise be lost if the company provided insurance to the wrong people.

Objective

The objective is to predict the life expectancy with the help of different factors affecting the life expectancy by establishing a relationship between the different variables.

Solution Approach

Since we have to predict the life expectancy and the life expectancy has continuous value, we can use regression model(OLS) to predict the target variable.

Data Description

For different countries and years, the dataset contains immunization factors, mortality factors, economic factors, social factors, and other health-related factors. The insurance company compiled the data from many countries in order to determine life expectancy and determine what factors influence life expectancy.

Data Dictionary

- Country: Country
- Year: Year
- Status: Developed or Developing status
- Life expectancy: Life Expectancy in years
- Adult Mortality: Adult Mortality Rates of both sexes (probability of dying between 15 and 60 years per 1000 population)
- Infant deaths: Number of Infant Deaths per 1000 population
- Alcohol: Alcohol, recorded per capita (15+) consumption (in liters of pure alcohol)

- percentage expenditure: Expenditure on health as a percentage of Gross Domestic Product per capita(%)
- Hepatitis B: Hepatitis B (HepB) immunization coverage among 1-year-olds (%)
- Measles: number of reported cases of Measles per 1000 population
- BMI: Average Body Mass Index of the entire population
- under-five deaths: Number of under-five deaths per 1000 population
- Polio: Polio (Pol3) immunization coverage among 1-year-olds (%)
- Total expenditure: General government expenditure on health as a percentage of total government expenditure (%)
- Diphtheria: Diphtheria tetanus toxoid and pertussis (DTP3) immunization coverage among 1-year-olds (%)
- HIV/AIDS: Deaths per 1000 live births due to HIV/AIDS (0-4 years)
- GDP: Gross Domestic Product per capita (in USD)
- Population: Population of the country
- thinness 1-19 years: Prevalence of thinness among children and adolescents for Age 10 to 19 (%)
- thinness 5-9 years: Prevalence of thinness among children for Age 5 to 9(%)
- Income composition of resources: Human Development Index in terms of income composition of resources (index ranging from 0 to 1)
- Schooling: Number of years of schooling

Importing necessary libraries

In [1]:
```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# split the data into train and test
from sklearn.model_selection import train_test_split

# to build linear regression_model using statsmodel
import statsmodels.api as sm

# to build linear regression_model using sklearn
from sklearn.linear_model import LinearRegression

# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [2]:
```
# Loading the dataset
data = pd.read_csv("./Life_Expectancy_Data.csv")
```

Data Overview

In [3]:
```
data.head()
```

Out[3]:

	Country	Year	Status	Life expectancy	Adult Mortality	Infant deaths	Alcohol	Percentage expenditure	Hepati
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	6
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	6
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	6
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	6
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	6

5 rows × 22 columns

Observations

- The life expectancy varies from 59.2 to 65 years.
- The *Status* column seems to have text values, which will have to be converted to numerics for modeling purposes.

In [8]:
```
# number of rows and columns
print("There are {0} rows and {1} columns in the dataset".format(data.shape[0],
```

```
There are 2938 rows and 22 columns in the dataset
```

In [9]:
```
# column datatypes and number of non-null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Country                         2938 non-null   object 
 1   Year                            2938 non-null   int64  
 2   Status                          2938 non-null   object 
 3   Life expectancy                 2928 non-null   float64
 4   Adult Mortality                 2928 non-null   float64
 5   Infant deaths                   2938 non-null   int64  
 6   Alcohol                         2744 non-null   float64
 7   Percentage expenditure          2938 non-null   float64
 8   Hepatitis B                     2385 non-null   float64
 9   Measles                         2938 non-null   int64  
 10  BMI                             2904 non-null   float64
 11  Under-five deaths               2938 non-null   int64  
 12  Polio                           2919 non-null   float64
 13  Total expenditure               2712 non-null   float64
 14  Diphtheria                      2919 non-null   float64
 15  HIV/AIDS                        2938 non-null   float64
 16  GDP                             2490 non-null   float64
 17  Population                      2286 non-null   float64
 18  Thinness  1-19 years            2904 non-null   float64
 19  Thinness 5-9 years              2904 non-null   float64
 20  Income composition of resources 2771 non-null   float64
 21  Schooling                       2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

Observations

- Most of the columns in the data are numeric in nature (integer or float).
- `Country` and `Status` columns are of *object* type.
- Some columns seem to have null (or missing) values too.

In [31]:
```
# Let's look at the statistical summary of the data
data.describe(include='all').T
```

Out[31]:

	count	unique	top	freq	mean	std	min	
Country	2938	193	Afghanistan	16	NaN	NaN	NaN	
Year	2938.0	NaN	NaN	NaN	2007.51872	4.613841	2000.0	
Status	2938	2	Developing	2426	NaN	NaN	NaN	
Life expectancy	2928.0	NaN	NaN	NaN	69.224932	9.523867	36.3	
Adult Mortality	2928.0	NaN	NaN	NaN	164.796448	124.292079	1.0	
Infant deaths	2938.0	NaN	NaN	NaN	30.303948	117.926501	0.0	
Alcohol	2744.0	NaN	NaN	NaN	4.602861	4.052413	0.01	
Percentage expenditure	2938.0	NaN	NaN	NaN	738.251295	1987.914858	0.0	
Hepatitis B	2385.0	NaN	NaN	NaN	80.940461	25.070016	1.0	
Measles	2938.0	NaN	NaN	NaN	2419.59224	11467.272489	0.0	
BMI	2904.0	NaN	NaN	NaN	38.321247	20.044034	1.0	
Under-five deaths	2938.0	NaN	NaN	NaN	42.035739	160.445548	0.0	
Polio	2919.0	NaN	NaN	NaN	82.550188	23.428046	3.0	
Total expenditure	2712.0	NaN	NaN	NaN	5.93819	2.49832	0.37	
Diphtheria	2919.0	NaN	NaN	NaN	82.324084	23.716912	2.0	
HIV/AIDS	2938.0	NaN	NaN	NaN	1.742103	5.077785	0.1	
GDP	2490.0	NaN	NaN	NaN	7483.158469	14270.169342	1.68135	4
Population	2286.0	NaN	NaN	NaN	12753375.120052	61012096.508428	34.0	
Thinness 1-19 years	2904.0	NaN	NaN	NaN	4.839704	4.420195	0.1	
Thinness 5-9 years	2904.0	NaN	NaN	NaN	4.870317	4.508882	0.1	
Income composition of resources	2771.0	NaN	NaN	NaN	0.627551	0.210904	0.0	
Schooling	2775.0	NaN	NaN	NaN	11.992793	3.35892	0.0	

Observations

- There are 193 countries.
- Most of the countries in the dataset are developing countries.
- The average life expectancy is ~69 years.

missing values.

- We will drop the missing values in the target variable (`Life expectancy`).
- We will replace the missing values in each column with its median for the predictor variables.

In [32]:
```
# create a copy of the original data
df = data.copy()
```

In [33]:
```
# dropping missing values in the target
df.dropna(subset=["Life expectancy"], inplace=True)
```

In [34]:
```
# filling missing values using the column median for the predictor variables
medianFiller = lambda x: x.fillna(x.median())
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()
df[numeric_columns] = df[numeric_columns].apply(medianFiller, axis=0)
```

In [35]:
```
# checking the number of missing values
df.isnull().sum()
```

Out[35]:
```
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
dtype: int64
```

- All the missing values have been treated.

In [36]:
```
# check the number of unique values in each column of the dataframe
df.nunique()
```

```
Out[36]: Country                            183
         Year                                16
         Status                               2
         Life expectancy                    362
         Adult Mortality                    425
         Infant deaths                      209
         Alcohol                           1076
         Percentage expenditure            2323
         Hepatitis B                         87
         Measles                            958
         BMI                                603
         Under-five deaths                  252
         Polio                               73
         Total expenditure                  816
         Diphtheria                          81
         HIV/AIDS                           200
         GDP                               2485
         Population                        2278
         Thinness  1-19 years               200
         Thinness 5-9 years                 207
         Income composition of resources    625
         Schooling                          173
         dtype: int64
```

Observations

- The *Status* column has 2 unique values.
- The *Country* column has 183 unique values.

Graphical visualization of the data to understand it in a better way.

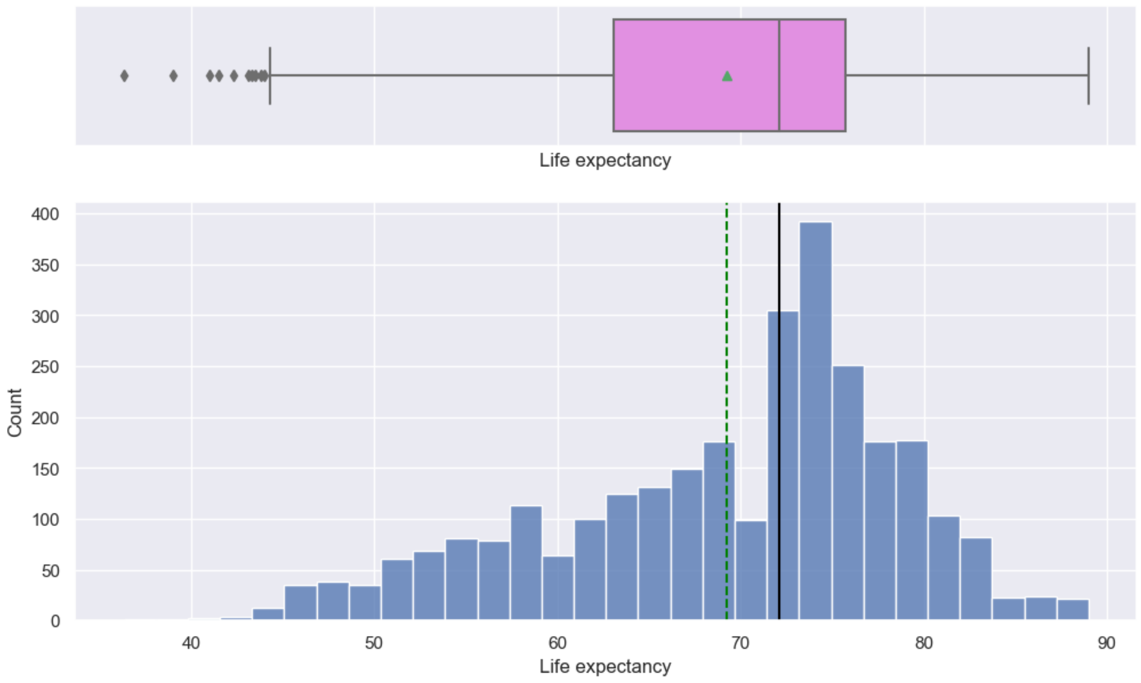
EDA

Univariate analysis

```
In [41]: # function to plot a boxplot and a histogram along the same scale.


         def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
             """
             Boxplot and histogram combined

             data: dataframe
             feature: dataframe column
             figsize: size of figure (default (12,7))
             kde: whether to the show density curve (default False)
             bins: number of bins for histogram (default None)
             """
             f2, (ax_box2, ax_hist2) = plt.subplots(
                 nrows=2,  # Number of rows of the subplot grid= 2
                 sharex=True,  # x-axis will be shared among all subplots
                 gridspec_kw={"height_ratios": (0.25, 0.75)},
                 figsize=figsize,
             )  # creating the 2 subplots
             sns.boxplot(
                 data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
```

```
    )  # boxplot will be created and a star will indicate the mean value of the
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [42]:
```
histogram_boxplot(df, "Life expectancy")
```

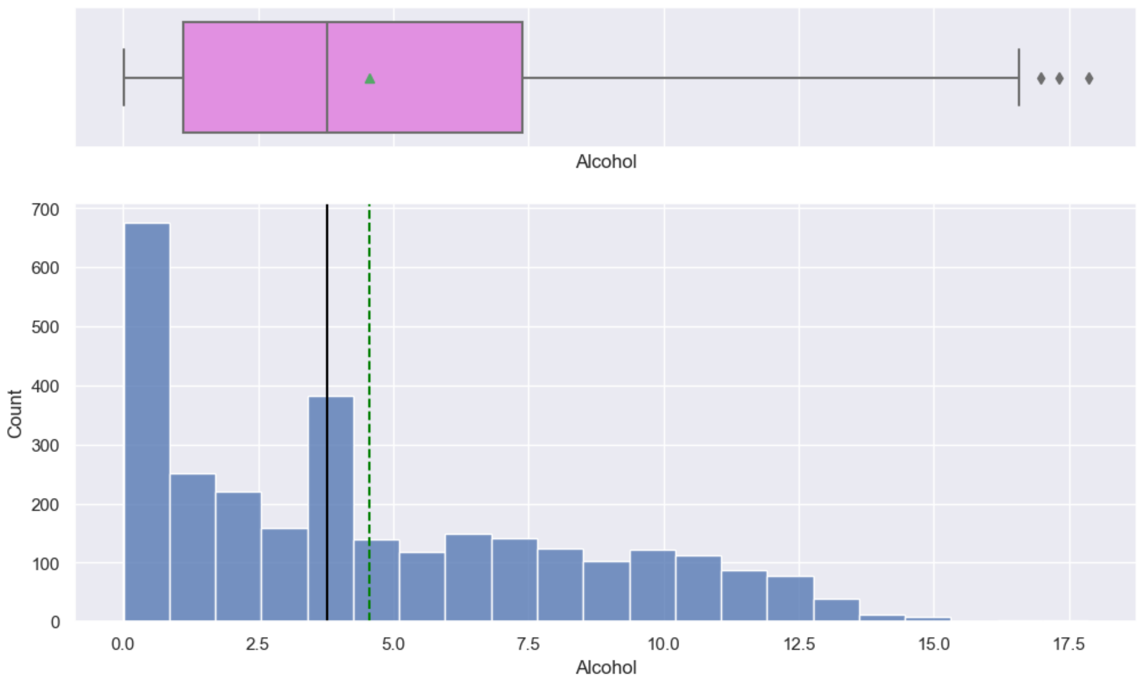


Observations

- `Life expectancy` is left-skewed, which means some countries have life expectancy less than 45 years.
- Mean life expectancy is around 72 years.

per capita alcohol consumption

In [43]:
```
histogram_boxplot(df, "Alcohol")
```

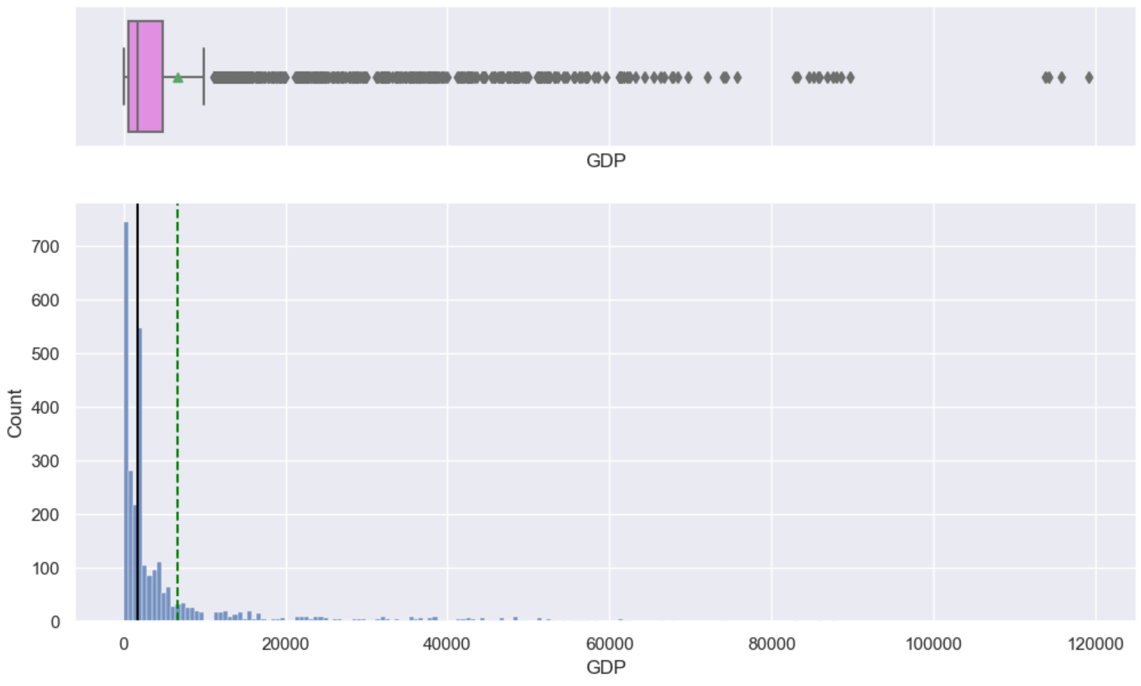


Observations

- The median alcohol consumption is 3.75 litres.
- There are outliers where average alcohol consumption is more than 15 litres.
- The distribution is right-skewed.

Let's explore GDP

In [49]:
```
histogram_boxplot(df, "GDP")
```

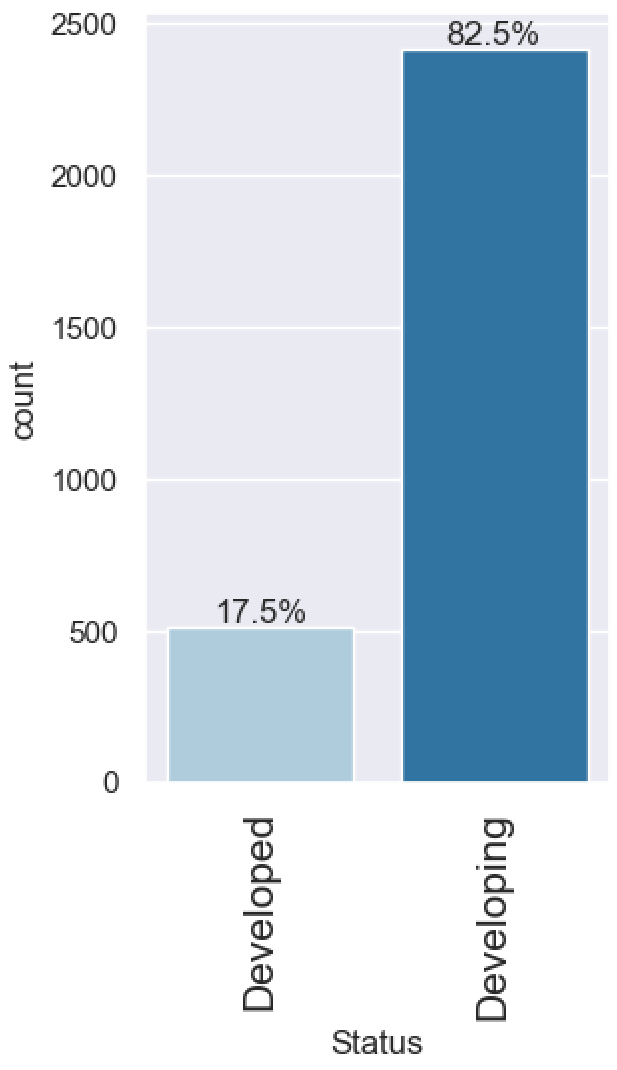


Observations

- The distribution of GDP is heavily skewed to the right.
- The outliers to the right indicate that many countries have a very high GDP.

In [50]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all le
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

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

In [51]:
```
labeled_barplot(df, "Status", perc=True)
```



- More than 80% of the countries in the data are developing countries.

Bivariate Analysis

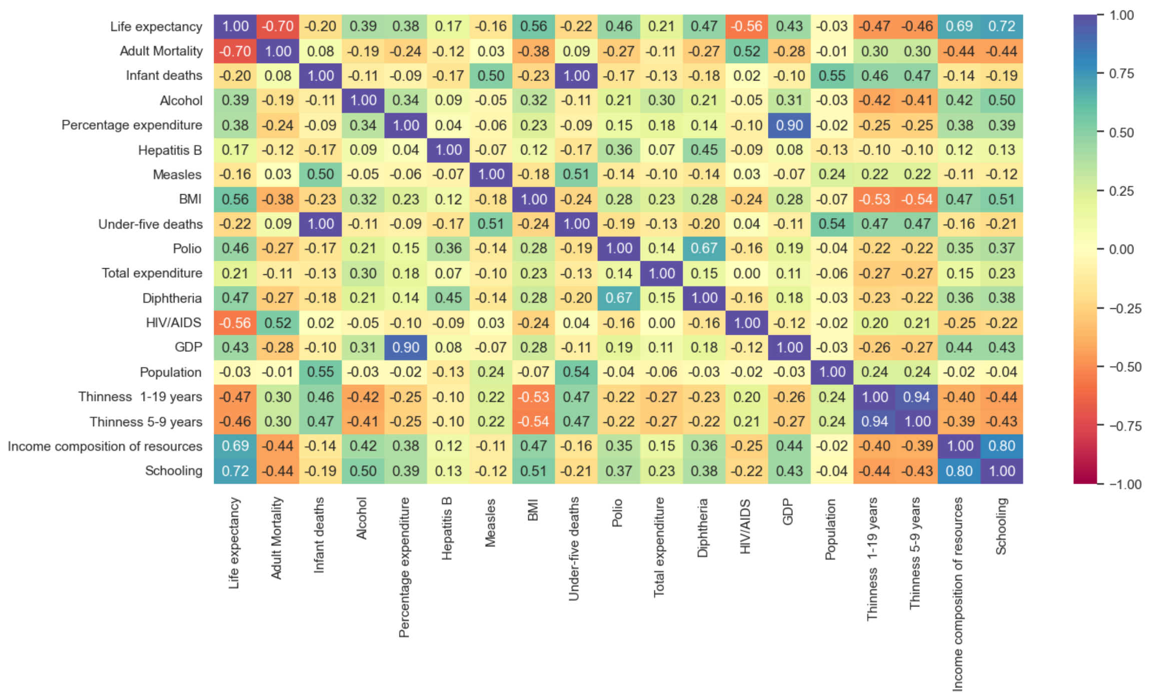
correlations.

In [52]:
```
# correlation of all attributes with life expectancy
df[df.columns[:]].corr()["Life expectancy"][:]
```

```
C:\Users\pavanksu2009\AppData\Local\Temp\ipykernel_9716\492482843.py:2: FutureW
arning: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  df[df.columns[:]].corr()["Life expectancy"][:]
```

```
Out[52]: Year                                0.170033
         Life expectancy                     1.000000
         Adult Mortality                    -0.696359
         Infant deaths                      -0.196557
         Alcohol                             0.390674
         Percentage expenditure              0.381864
         Hepatitis B                         0.171255
         Measles                            -0.157586
         BMI                                 0.558888
         Under-five deaths                  -0.222529
         Polio                               0.459458
         Total expenditure                   0.209588
         Diphtheria                          0.473268
         HIV/AIDS                           -0.556556
         GDP                                 0.430991
         Population                         -0.028842
         Thinness  1-19 years               -0.467859
         Thinness 5-9 years                 -0.462645
         Income composition of resources     0.688591
         Schooling                           0.717314
         Name: Life expectancy, dtype: float64
```

```
In [53]: numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
         numeric_columns.remove("Year")  # dropping year column as it is temporal variabl

         # correlation heatmap
         plt.figure(figsize=(15, 7))
         sns.heatmap(
             df[numeric_columns].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Sp
         )
         plt.show()
```



Observations

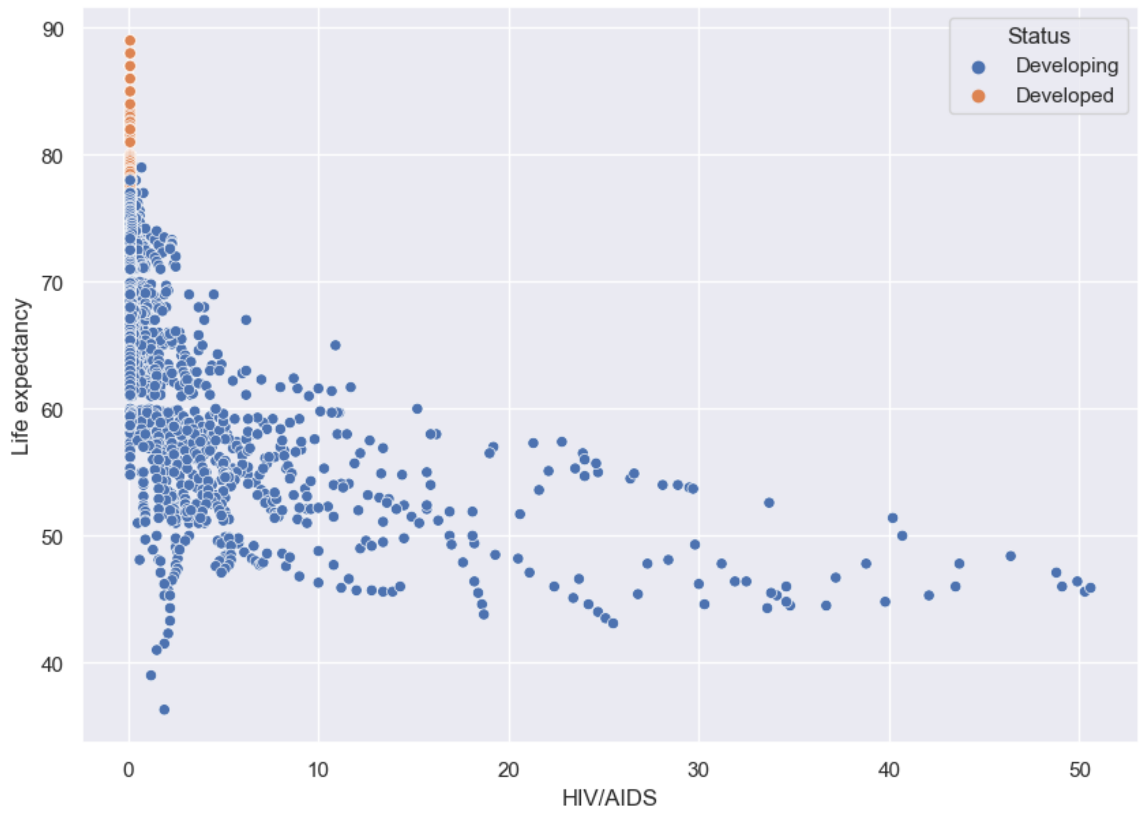
- `Life expectancy` is highly negatively correlated with `Adult Mortality` and `HIV/AIDs`, which means that as adult mortality and HIV death (0-4 years) increases, life expectancy tends to decrease.

- `Life expectancy` is highly positively correlated with `Schooling` and `Income composition of resources`, which means that as schooling years of citizens in a country and income composition of resources increases, life expectancy tends to increase.

graphs of a few variables that are highly correlated with `Life expectancy`.

`Life expectancy` vs `HIV/AIDS` vs `Status`

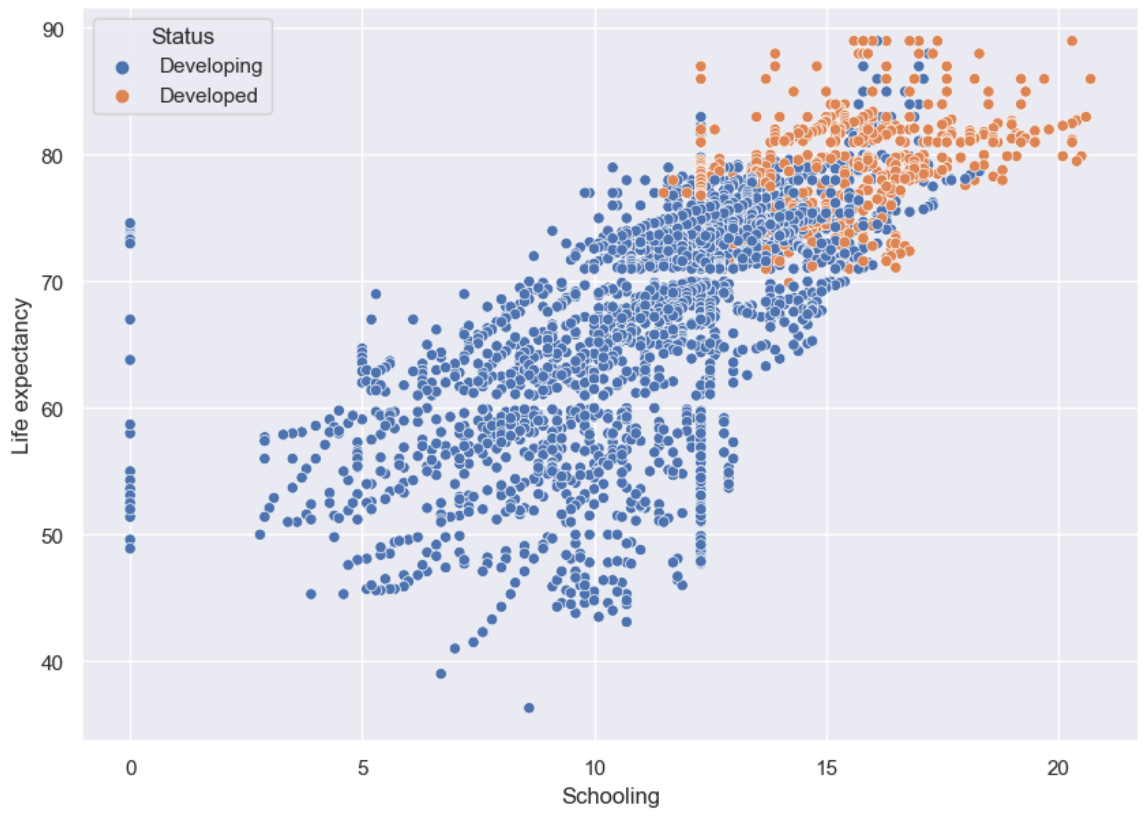
In [54]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Life expectancy", x="HIV/AIDS", hue="Status", data=df)
plt.show()
```



- Developed countries have very low cases of HIV/AIDS.

`Life expectancy` vs `Schooling` vs `Status`

In [55]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Life expectancy", x="Schooling", hue="Status", data=df)
plt.show()
```



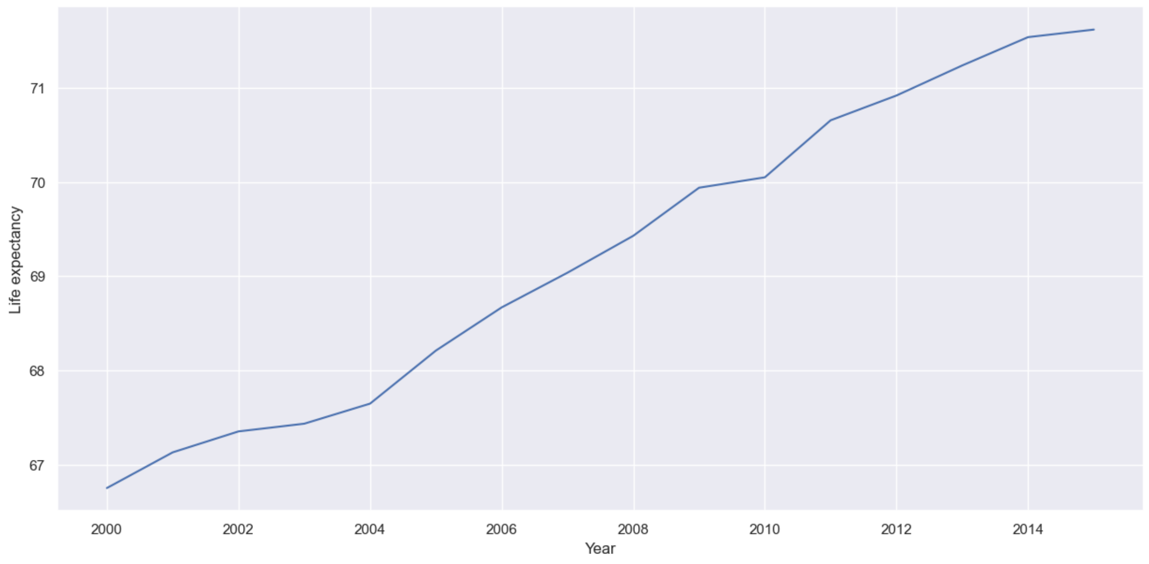
Observations

- Majority of the developed countries have schooling of more than 13 years.
- Developing countries have a higher variance in schooling years.

Let's check the variation in life expectancy across years.

```
In [56]: # average life expectancy over the years
plt.figure(figsize=(15, 7))
sns.lineplot(x="Year", y="Life expectancy", data=df, ci=None)
plt.show()
```

```
C:\Users\pavanksu2009\AppData\Local\Temp\ipykernel_9716\1913134968.py:3: Future
Warning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(x="Year", y="Life expectancy", data=df, ci=None)
```



- Overall life expectancy of the world population is increasing over the years.

In [57]:
```
pip install pycountry-convert
```

```
Requirement already satisfied: pycountry-convert in c:\users\pavanksu2009\.virt
ualenvs\system32-zwnxhztr\lib\site-packages (0.7.2)
Requirement already satisfied: pytest>=3.4.0 in c:\users\pavanksu2009\.virtuale
nvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (7.2.0)
Requirement already satisfied: pytest-mock>=1.6.3 in c:\users\pavanksu2009\.vir
tualenvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (3.10.0)
Requirement already satisfied: pprintpp>=0.3.0 in c:\users\pavanksu2009\.virtua
lenvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (0.4.0)
Requirement already satisfied: repoze.lru>=0.7 in c:\users\pavanksu2009\.virtua
lenvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (0.7)
Requirement already satisfied: pycountry>=16.11.27.1 in c:\users\pavanksu2009\.
virtualenvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (22.3.
5)
Requirement already satisfied: pytest-cov>=2.5.1 in c:\users\pavanksu2009\.virt
ualenvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (4.0.0)
Requirement already satisfied: wheel>=0.30.0 in c:\users\pavanksu2009\.virtuale
nvs\system32-zwnxhztr\lib\site-packages (from pycountry-convert) (0.37.1)
Requirement already satisfied: setuptools in c:\users\pavanksu2009\.virtualenvs
\system32-zwnxhztr\lib\site-packages (from pycountry>=16.11.27.1->pycountry-con
vert) (65.3.0)
Requirement already satisfied: attrs>=19.2.0 in c:\users\pavanksu2009\.virtuale
nvs\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-convert)
(22.1.0)
Requirement already satisfied: iniconfig in c:\users\pavanksu2009\.virtualenvs
\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-convert)
(1.1.1)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in c:\users\pavanksu200
9\.virtualenvs\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycount
ry-convert) (1.0.4)
Requirement already satisfied: tomli>=1.0.0 in c:\users\pavanksu2009\.virtualen
vs\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-convert)
(2.0.1)
Requirement already satisfied: colorama in c:\users\pavanksu2009\.virtualenvs\s
ystem32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-convert) (0.
4.6)
Requirement already satisfied: packaging in c:\users\pavanksu2009\.virtualenvs
\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-convert) (2
1.3)
Requirement already satisfied: pluggy<2.0,>=0.12 in c:\users\pavanksu2009\.virt
ualenvs\system32-zwnxhztr\lib\site-packages (from pytest>=3.4.0->pycountry-conv
ert) (1.0.0)
Requirement already satisfied: coverage[toml]>=5.2.1 in c:\users\pavanksu2009\.
virtualenvs\system32-zwnxhztr\lib\site-packages (from pytest-cov>=2.5.1->pycoun
try-convert) (6.5.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\pavanksu200
9\.virtualenvs\system32-zwnxhztr\lib\site-packages (from packaging->pytest>=3.
4.0->pycountry-convert) (3.0.9)
Note: you may need to restart the kernel to use updated packages.
```

```
[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [58]:

```python
# Let's group coutries into continents
import pycountry_convert as pc


def country_to_continent(country_name):
    """
    country_name : name of country for which continent is needed
    """
    if "(" in country_name:
        country_name = country_name.split(" ")[0]
    country_alpha2 = pc.country_name_to_country_alpha2(country_name)
    country_continent_code = pc.country_alpha2_to_continent_code(country_alpha2)
    country_continent_name = pc.convert_continent_code_to_continent_name(
        country_continent_code
    )
    return country_continent_name
```

- Above error is arising because names of the countries are different from what the library has.
- In order to resolve this, we looked at all country names that caused this error and hard-coded them as shown below.

In [121…

```
loc = df.Country.tolist()
continent = dict()

# hard-coding the continent names of those countries which were giving error wit
for cn in loc:
    if cn == "Republic of Korea":
        continent[cn] = "Asia"
    elif cn == "The former Yugoslav republic of Macedonia":
        continent[cn] = "Europe"
    elif cn == "Timor-Leste":
        continent[cn] = "Asia"
    else:
        continent[cn] = country_to_continent(cn)
```

In [122…

```
# mapping every country to its continent
df["Continent"] = df["Country"].map(continent)
```

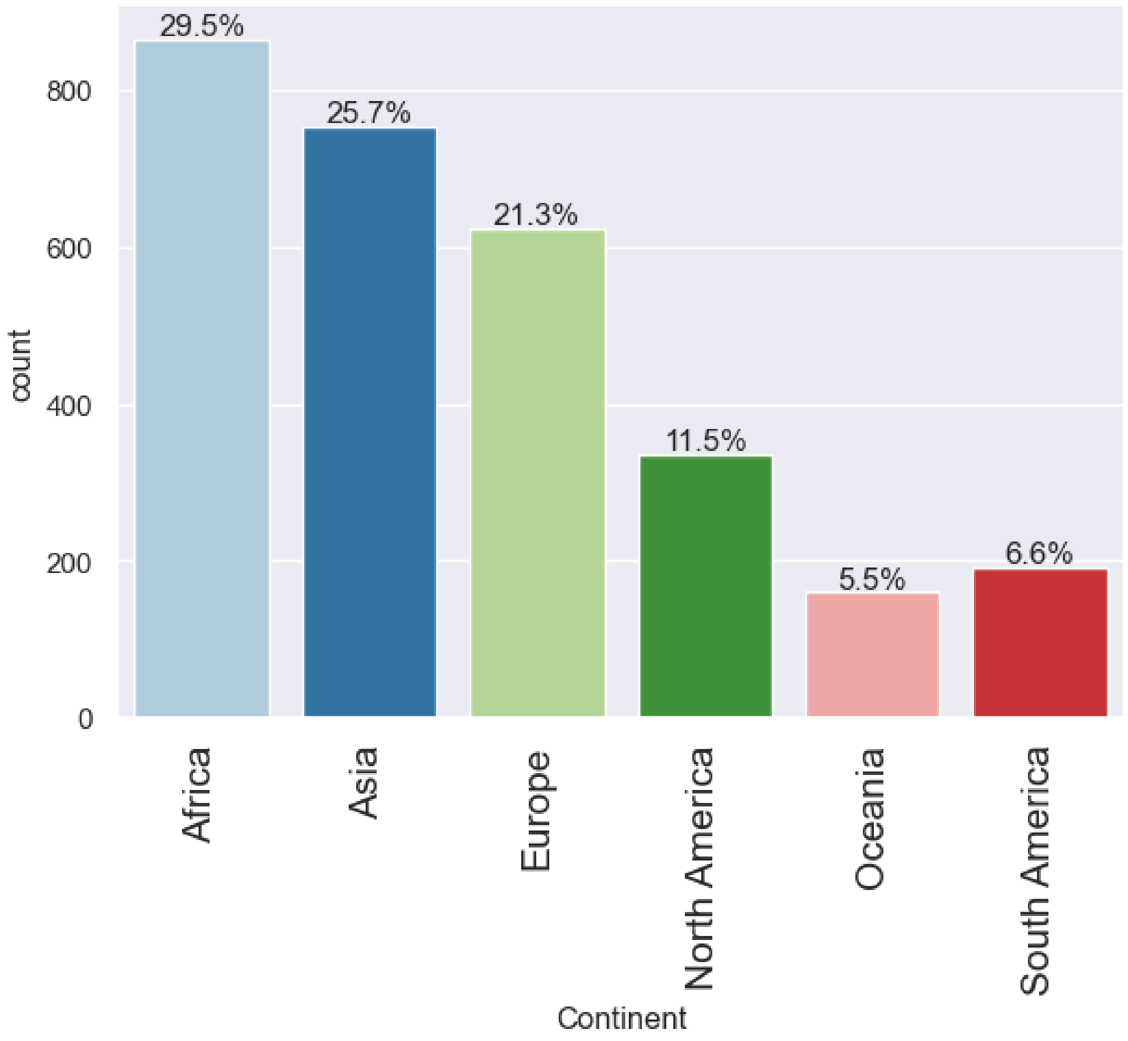
In [123…

```
# let us look at unique continents
print(df["Country"].map(continent).unique())
```

```
['Asia' 'Europe' 'Africa' 'North America' 'South America' 'Oceania']
```

In [124…

```
labeled_barplot(df, "Continent", perc=True)
```

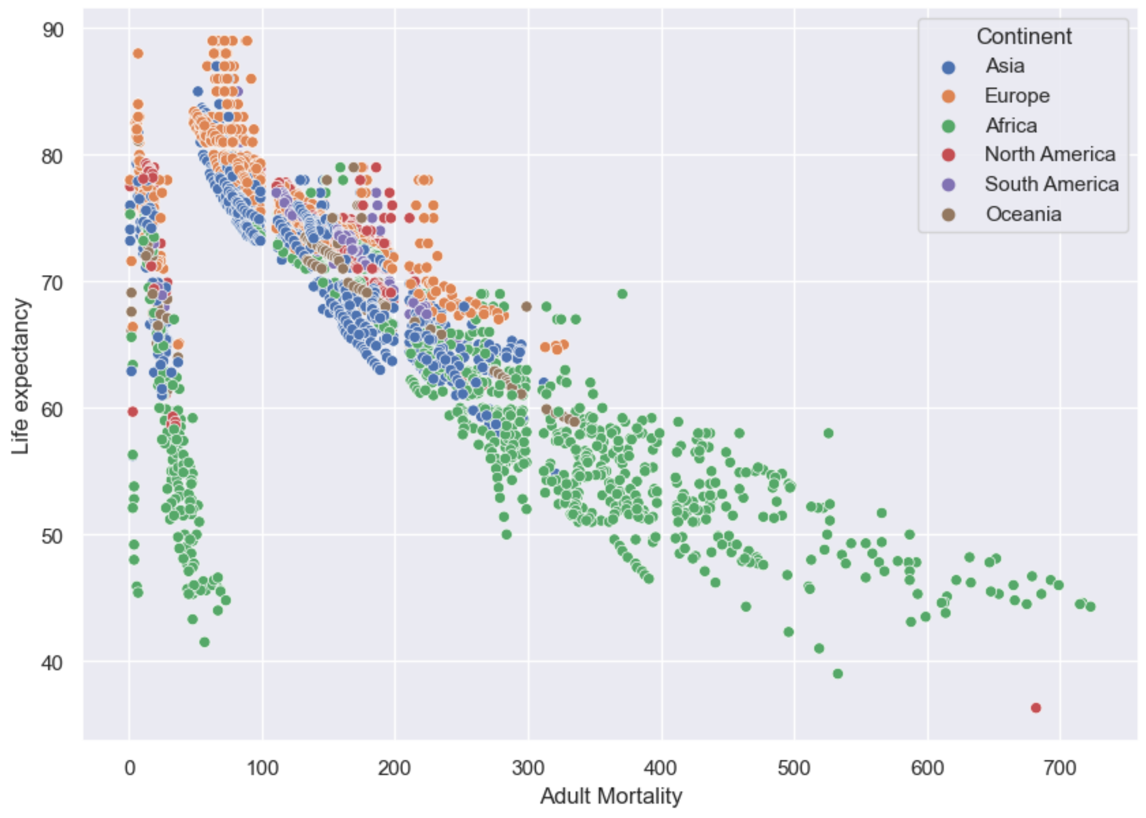


Observations

- More than 75% of the data points are from Africa, Asia, and Europe.
- Oceania accounts for only 5.5% of the data points.

`Life expectancy` vs `Adult Mortality` vs `Continent`

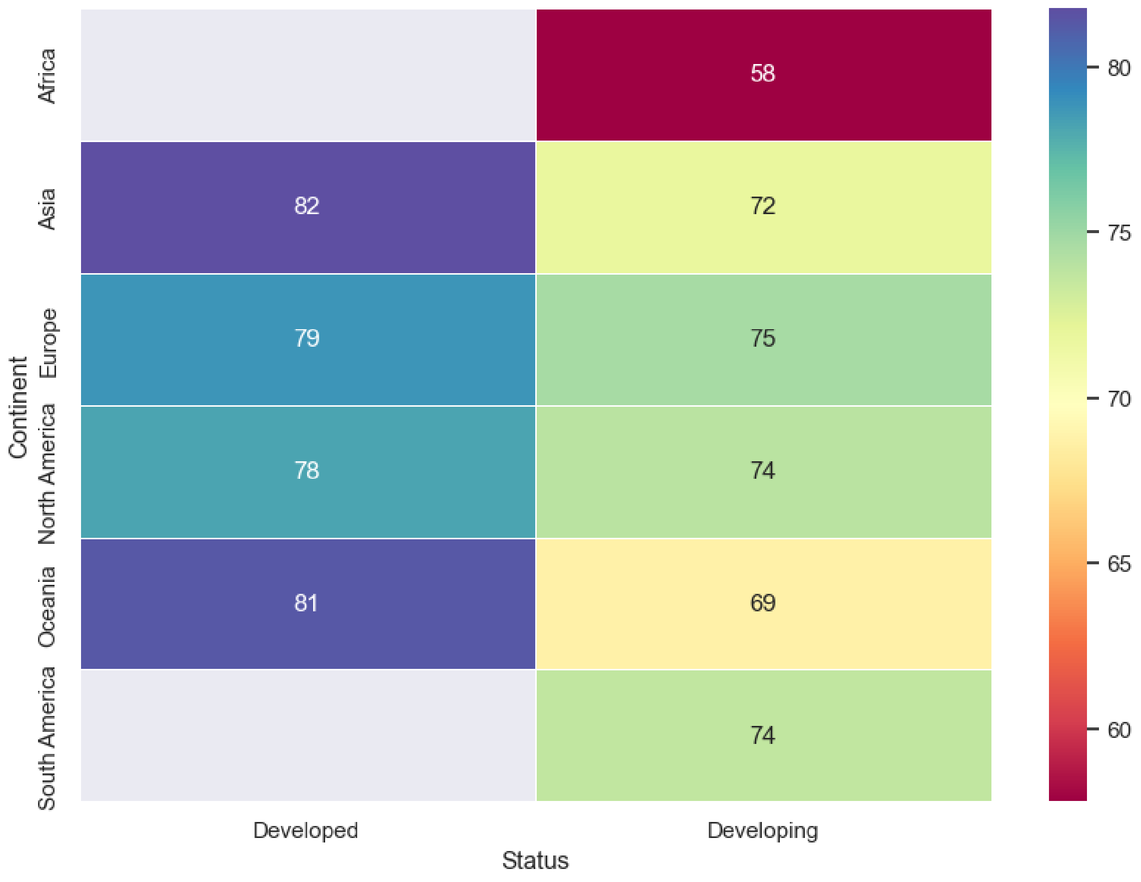
In [65]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Life expectancy", x="Adult Mortality", hue="Continent", data=
plt.show()
```



- Many European countries have had life expectancy higher than 80 years for some years.
- Most of the African countries have higher adult mortality and life expectancy lower than 65 years.

Median `Life expectancy` by `Country` and `Status`

In [66]:
```
df_hm = df.pivot_table(
    index="Continent", columns="Status", values="Life expectancy", aggfunc=np.me
)

# Draw a heatmap
f, ax = plt.subplots(figsize=(10, 7))
sns.heatmap(df_hm, cmap="Spectral", linewidths=0.5, annot=True, ax=ax)
plt.show()
```



- Developed countries from Asia have the highest life expectancy.

convert the *object* type columns to *category* type

In [67]:
```
df["Country"] = df["Country"].astype("category")
df["Status"] = df["Status"].astype("category")
df["Continent"] = df["Continent"].astype("category")
```

Linear Model Building

1. We want to predict the life expectancy.
2. Before we proceed to build a model, we'll have to encode categorical features.
3. We'll split the data into train and test to be able to evaluate the model that we build on the train data.
4. We will build a Linear Regression model using the train data and then check it's performance.

In [68]:
```
# defining X and y variables
X = df.drop(["Life expectancy", "Country"], axis=1)
y = df["Life expectancy"]

print(X.head())
print(y.head())
```

```
   Year      Status  Adult Mortality  Infant deaths  Alcohol  \
0  2015  Developing            263.0             62     0.01
1  2014  Developing            271.0             64     0.01
2  2013  Developing            268.0             66     0.01
3  2012  Developing            272.0             69     0.01
4  2011  Developing            275.0             71     0.01

   Percentage expenditure  Hepatitis B  Measles   BMI  Under-five deaths  ...
\
0               71.279624         65.0     1154  19.1                 83  ...
1               73.523582         62.0      492  18.6                 86  ...
2               73.219243         64.0      430  18.1                 89  ...
3               78.184215         67.0     2787  17.6                 93  ...
4                7.097109         68.0     3013  17.2                 97  ...

   Total expenditure  Diphtheria  HIV/AIDS         GDP  Population  \
0               8.16        65.0       0.1  584.259210  33736494.0
1               8.18        62.0       0.1  612.696514    327582.0
2               8.13        64.0       0.1  631.744976  31731688.0
3               8.52        67.0       0.1  669.959000   3696958.0
4               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  Continent
0       10.1       Asia
1       10.0       Asia
2        9.9       Asia
3        9.8       Asia
4        9.5       Asia

[5 rows x 21 columns]
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy, dtype: float64
```

In [69]:
```python
# let's add the intercept to data
X = sm.add_constant(X)
```

In [70]:
```python
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolist(),
    drop_first=True,
)
X.head()
```

Out[70]:

	const	Year	Adult Mortality	Infant deaths	Alcohol	Percentage expenditure	Hepatitis B	Measles	BMI	Under-five deaths	...
0	1.0	2015	263.0	62	0.01	71.279624	65.0	1154	19.1	83	...
1	1.0	2014	271.0	64	0.01	73.523582	62.0	492	18.6	86	...
2	1.0	2013	268.0	66	0.01	73.219243	64.0	430	18.1	89	...
3	1.0	2012	272.0	69	0.01	78.184215	67.0	2787	17.6	93	...
4	1.0	2011	275.0	71	0.01	7.097109	68.0	3013	17.2	97	...

5 rows × 26 columns

In [71]:
```
# splitting the data in 70:30 ratio for train to test data

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [72]:
```
print("Number of rows in train data =", x_train.shape[0])
print("Number of rows in test data =", x_test.shape[0])
```

```
Number of rows in train data = 2049
Number of rows in test data = 879
```

In [73]:
```
olsmodel = sm.OLS(y_train, x_train).fit()
print(olsmodel.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Life expectancy   R-squared:                       0.847
Model:                            OLS   Adj. R-squared:                  0.845
Method:                 Least Squares   F-statistic:                     448.3
Date:                Thu, 01 Dec 2022   Prob (F-statistic):               0.00
Time:                        15:34:07   Log-Likelihood:                -5612.6
No. Observations:                2049   AIC:                         1.128e+04
Df Residuals:                    2023   BIC:                         1.142e+04
Df Model:                          25
Covariance Type:            nonrobust
===============================================================================
====================
                                      coef    std err          t      P>|t|
[0.025      0.975]
-------------------------------------------------------------------------------
--------------------
const                             -10.0359     39.682     -0.253      0.800
-87.858      67.786
Year                                0.0334      0.020      1.683      0.093
-0.006       0.072
Adult Mortality                    -0.0162      0.001    -17.946      0.000
-0.018      -0.014
Infant deaths                       0.0639      0.010      6.511      0.000
0.045       0.083
Alcohol                            -0.0505      0.033     -1.508      0.132
-0.116       0.015
Percentage expenditure              0.0002   9.93e-05      1.695      0.090   -
2.64e-05       0.000
Hepatitis B                        -0.0145      0.004     -3.393      0.001
-0.023      -0.006
Measles                         -1.559e-05   9.27e-06     -1.682      0.093   -
3.38e-05    2.58e-06
BMI                                 0.0339      0.006      5.772      0.000
0.022       0.045
Under-five deaths                  -0.0485      0.007     -6.750      0.000
-0.063      -0.034
Polio                               0.0323      0.005      6.551      0.000
0.023       0.042
Total expenditure                  -0.0060      0.039     -0.152      0.879
-0.083       0.071
Diphtheria                          0.0309      0.005      6.014      0.000
0.021       0.041
HIV/AIDS                           -0.3832      0.020    -18.797      0.000
-0.423      -0.343
GDP                              2.496e-05     1.5e-05      1.663      0.096   -
4.47e-06    5.44e-05
Population                       3.075e-10   2.31e-09      0.133      0.894   -
4.22e-09    4.84e-09
Thinness  1-19 years               -0.0236      0.061     -0.385      0.700
-0.143       0.096
Thinness 5-9 years                 -0.0729      0.061     -1.198      0.231
-0.192       0.046
Income composition of resources     4.0795      0.721      5.658      0.000
2.665       5.494
Schooling                           0.6186      0.048     12.880      0.000
0.524       0.713
Status_Developing                  -2.5795      0.352     -7.318      0.000
-3.271      -1.888
Continent_Asia                      4.2862      0.287     14.922      0.000
```

```
3.723       4.850
Continent_Europe                    3.9647      0.413      9.597      0.000
3.155       4.775
Continent_North America             5.8618      0.366     16.031      0.000
5.145       6.579
Continent_Oceania                   2.4254      0.459      5.286      0.000
1.526       3.325
Continent_South America             3.9444      0.445      8.860      0.000
3.071       4.817
==============================================================================
Omnibus:                       79.914   Durbin-Watson:                   2.016
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              212.485
Skew:                          -0.142   Prob(JB):                     7.24e-47
Kurtosis:                       4.552   Cond. No.                     2.09e+10
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.09e+10. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the Regression Results:

1. **Adjusted. R-squared**: It reflects the fit of the model.

   - Adjusted R-squared values generally range from 0 to 1, where a higher value generally indicates a better fit, assuming certain conditions are met.
   - In our case, the value for adj. R-squared is **0.845**, which is good!
2. ***const* coefficient**: It is the Y-intercept.

   - It means that if all the predictor variable coefficients are zero, then the expected output (i.e., Y) would be equal to the *const* coefficient.
   - In our case, the value for *const* coefficient is **-10.0359**
3. **Coefficient of a predictor variable**: It represents the change in the output Y due to a change in the predictor variable (everything else held constant).

   - In our case, the coefficient of `Adult Mortality` is -0.0162.

Let's check the performance of the model using different metrics.

- We will be using metric functions defined in sklearn for RMSE, MAE, and R^2.
- We will define a function to calculate MAPE and adjusted R^2.
  - The mean absolute percentage error (MAPE) measures the accuracy of predictions as a percentage, and can be calculated as the average absolute percent error for each predicted value minus actual values divided by actual values. It works best if there are no extreme values in the data and none of the actual values are 0.
- We will create a function which will print out all the above metrics in one go.

In [74]:
```
# function to compute adjusted R-squared
```

```
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


# function to compute MAPE
def mape_score(targets, predictions):
    return np.mean(np.abs(targets - predictions) / targets) * 100


# function to compute different metrics to check performance of a regression mod
def model_performance_regression(model, predictors, target):
    """
    Function to compute different metrics to check regression model performance

    model: regressor
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    r2 = r2_score(target, pred)  # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)  # to compute adjusted R-squa
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)  # to compute MAE
    mape = mape_score(target, pred)  # to compute MAPE

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "RMSE": rmse,
            "MAE": mae,
            "R-squared": r2,
            "Adj. R-squared": adjr2,
            "MAPE": mape,
        },
        index=[0],
    )

    return df_perf
```

In [75]:
```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel_train_perf = model_performance_regression(olsmodel, x_train, y_train)
olsmodel_train_perf
```

```
Training Performance
```

Out[75]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.7444	2.832731	0.847098	0.845132	4.308428

In [76]:
```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
```

```
olsmodel_test_perf = model_performance_regression(olsmodel, x_test, y_test)
olsmodel_test_perf
```

```
Test Performance
```

Out[76]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.730845	2.820497	0.842311	0.837499	4.280578

Observations

- The training R^2 is 0.85, so the model is not underfitting.
- The train and test RMSE and MAE are comparable, so the model is not overfitting either.
- MAE suggests that the model can predict life expectancy within a mean error of 2.8 years on the test data.
- MAPE of 4.3 on the test data means that we are able to predict within 4.3% of the life expectancy.

Checking Linear Regression Assumptions

We will be checking the following Linear Regression assumptions:

1. **No Multicollinearity**
2. **Linearity of variables**
3. **Independence of error terms**
4. **Normality of error terms**
5. **No Heteroscedasticity**

TEST FOR MULTICOLLINEARITY

- Multicollinearity occurs when predictor variables in a regression model are correlated. This correlation is a problem because predictor variables should be independent. If the correlation between variables is high, it can cause problems when we fit the model and interpret the results. When we have multicollinearity in the linear model, the coefficients that the model suggests are unreliable.
- There are different ways of detecting (or testing) multicollinearity. One such way is by using the Variance Inflation Factor, or VIF.
- **Variance Inflation Factor (VIF)**: Variance inflation factors measure the inflation in the variances of the regression parameter estimates due to collinearities that exist among the predictors. It is a measure of how much the variance of the estimated

regression coefficient β_k is "inflated" by the existence of correlation among the predictor variables in the model.

- If VIF is 1, then there is no correlation among the kth predictor and the remaining predictor variables, and hence, the variance of β_k is not inflated at all.

- **General Rule of thumb**:
  - If VIF is between 1 and 5, then there is low multicollinearity.
  - If VIF is between 5 and 10, we say there is moderate multicollinearity.
  - If VIF is exceeding 10, it shows signs of high multicollinearity.

In [77]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# we will define a function to check VIF
def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        variance_inflation_factor(predictors.values, i)
        for i in range(len(predictors.columns))
    ]
    return vif
```

In [78]:
```
checking_vif(x_train)
```

Out[78]:

	feature	VIF
0	const	227205.392829
1	Year	1.191704
2	Adult Mortality	1.835454
3	Infant deaths	191.008892
4	Alcohol	2.470353
5	Percentage expenditure	5.631140
6	Hepatitis B	1.322443
7	Measles	1.432634
8	BMI	1.921698
9	Under-five deaths	190.547440
10	Polio	1.909524
11	Total expenditure	1.250291
12	Diphtheria	2.107072
13	HIV/AIDS	1.569409
14	GDP	5.871180
15	Population	1.410645
16	Thinness 1-19 years	9.874031
17	Thinness 5-9 years	10.052169
18	Income composition of resources	3.179681
19	Schooling	3.534133
20	Status_Developing	2.590817
21	Continent_Asia	2.242959
22	Continent_Europe	4.110731
23	Continent_North America	2.009284
24	Continent_Oceania	1.635813
25	Continent_South America	1.625837

- `Thinness__1_19_years`, `Thinness__5_9_years`, `Percentage expenditure`, and `GDP` have VIFs slightly greater than 5.
- `Infant_deaths` and `Under_five_deaths` have a VIF score of much greater than 5.
  - Clearly these 2 variables are correlated with each other.
  - This does seem to make intuitive sense because the number of infant death and under-5 deaths would have a significant overlap.

Removing Multicollinearity

To remove multicollinearity

1. Drop every column one by one that has a VIF score greater than 5.
2. Look at the adjusted R-squared and RMSE of all these models.
3. Drop the variable that makes the least change in adjusted R-squared.
4. Check the VIF scores again.
5. Continue till you get all VIF scores under 5.

Let's define a function that will help us do this.

In [79]:
```
def treating_multicollinearity(predictors, target, high_vif_columns):
    """
    Checking the effect of dropping the columns showing high multicollinearity
    on model performance (adj. R-squared and RMSE)

    predictors: independent variables
    target: dependent variable
    high_vif_columns: columns having high VIF
    """
    # empty lists to store adj. R-squared and RMSE values
    adj_r2 = []
    rmse = []

    # build ols models by dropping one of the high VIF columns at a time
    # store the adjusted R-squared and RMSE in the lists defined previously
    for cols in high_vif_columns:
        # defining the new train set
        train = predictors.loc[:, ~predictors.columns.str.startswith(cols)]

        # create the model
        olsmodel = sm.OLS(target, train).fit()

        # adding adj. R-squared and RMSE to the lists
        adj_r2.append(olsmodel.rsquared_adj)
        rmse.append(np.sqrt(olsmodel.mse_resid))

    # creating a dataframe for the results
    temp = pd.DataFrame(
        {
            "col": high_vif_columns,
            "Adj. R-squared after_dropping col": adj_r2,
            "RMSE after dropping col": rmse,
        }
    ).sort_values(by="Adj. R-squared after_dropping col", ascending=False)
    temp.reset_index(drop=True, inplace=True)

    return temp
```

In [80]:
```
col_list = [
    "Infant deaths",
    "Under-five deaths",
    "Percentage expenditure",
    "GDP",
    "Thinness  1-19 years",
    "Thinness 5-9 years",
]
```

```
res = treating_multicollinearity(x_train, y_train, col_list)
res
```

Out[80]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	Thinness 1-19 years	0.845273	3.767592
1	Thinness 5-9 years	0.845175	3.768790
2	GDP	0.845073	3.770029
3	Percentage expenditure	0.845065	3.770129
4	Infant deaths	0.842042	3.806727
5	Under-five deaths	0.841800	3.809645

- Dropping `Under-five deaths` would have the maximum impact on the predictive power of the model (amongst the variables being considered).
- We'll drop `Thinness 1-19 years` and check the VIF again.

In [81]:

```
col_to_drop = "Thinness  1-19 years"
x_train2 = x_train.loc[:, ~x_train.columns.str.startswith(col_to_drop)]
x_test2 = x_test.loc[:, ~x_test.columns.str.startswith(col_to_drop)]

# Check VIF now
vif = checking_vif(x_train2)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  Thinness  1-19 years
```

Out[81]:

	feature	VIF
0	const	227163.382376
1	Year	1.191427
2	Adult Mortality	1.835454
3	Infant deaths	190.684749
4	Alcohol	2.465115
5	Percentage expenditure	5.631131
6	Hepatitis B	1.322159
7	Measles	1.432441
8	BMI	1.920484
9	Under-five deaths	190.336121
10	Polio	1.907730
11	Total expenditure	1.250287
12	Diphtheria	2.105441
13	HIV/AIDS	1.569208
14	GDP	5.870923
15	Population	1.410503
16	Thinness 5-9 years	2.298109
17	Income composition of resources	3.177579
18	Schooling	3.534092
19	Status_Developing	2.590815
20	Continent_Asia	2.239828
21	Continent_Europe	4.096344
22	Continent_North America	2.003104
23	Continent_Oceania	1.625101
24	Continent_South America	1.622022

- Dropping `Thinness 1-19 years` has brought the VIF of `Thinness 5-9 years` below 5.
- `Infant deaths` and `Under-five deaths` still have a VIF score of much greater than 5.

In [82]:
```
col_list = [
    "Infant deaths",
    "Under-five deaths",
    "Percentage expenditure",
    "GDP",
]
```

```
res = treating_multicollinearity(x_train2, y_train, col_list)
res
```

Out[82]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	GDP	0.845139	3.769228
1	Percentage expenditure	0.845130	3.769338
2	Infant deaths	0.842119	3.805800
3	Under-five deaths	0.841876	3.808728

- We will drop `GDP` next.

In [83]:

```
col_to_drop = "GDP"
x_train3 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test3 = x_test2.loc[:, ~x_test2.columns.str.startswith(col_to_drop)]

# Check VIF now
vif = checking_vif(x_train3)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  GDP
```

Out[83]:

	feature	VIF
0	const	224205.117775
1	Year	1.175926
2	Adult Mortality	1.833756
3	Infant deaths	190.073929
4	Alcohol	2.454605
5	Percentage expenditure	1.383310
6	Hepatitis B	1.321552
7	Measles	1.432193
8	BMI	1.914150
9	Under-five deaths	189.710454
10	Polio	1.905124
11	Total expenditure	1.228239
12	Diphtheria	2.105384
13	HIV/AIDS	1.566821
14	Population	1.410490
15	Thinness 5-9 years	2.289516
16	Income composition of resources	3.166265
17	Schooling	3.523918
18	Status_Developing	2.580709
19	Continent_Asia	2.221620
20	Continent_Europe	4.096308
21	Continent_North America	2.003063
22	Continent_Oceania	1.617673
23	Continent_South America	1.621238

- Dropping `GDP` has brought the VIF of `Percentage expenditure` below 5.
- `Infant deaths` and `Under-five deaths` still have a VIF score of much greater than 5.

In [84]:
```
col_list = [
    "Infant deaths",
    "Under-five deaths",
]

res = treating_multicollinearity(x_train3, y_train, col_list)
res
```

Out[84]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	Infant deaths	0.842069	3.806404
1	Under-five deaths	0.841830	3.809287

- We will drop `Infant deaths` next.

In [85]:
```
col_to_drop = "Infant deaths"
x_train4 = x_train3.loc[:, ~x_train3.columns.str.startswith(col_to_drop)]
x_test4 = x_test3.loc[:, ~x_test3.columns.str.startswith(col_to_drop)]

# Check VIF now
vif = checking_vif(x_train4)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  Infant deaths
```

Out[85]:

	feature	VIF
0	const	224187.636224
1	Year	1.175885
2	Adult Mortality	1.833601
3	Alcohol	2.426863
4	Percentage expenditure	1.382397
5	Hepatitis B	1.316589
6	Measles	1.432191
7	BMI	1.913549
8	Under-five deaths	2.219838
9	Polio	1.897319
10	Total expenditure	1.227935
11	Diphtheria	2.085718
12	HIV/AIDS	1.566659
13	Population	1.400031
14	Thinness 5-9 years	2.276925
15	Income composition of resources	3.149154
16	Schooling	3.523843
17	Status_Developing	2.580706
18	Continent_Asia	2.113664
19	Continent_Europe	4.016625
20	Continent_North America	1.955165
21	Continent_Oceania	1.592618
22	Continent_South America	1.584544

- **The above predictors have no multicollinearity and the assumption is satisfied.**
- **Let's check the model summary.**

In [86]:
```
olsmod1 = sm.OLS(y_train, x_train4).fit()
print(olsmod1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Life expectancy   R-squared:                       0.844
Model:                            OLS   Adj. R-squared:                  0.842
Method:                 Least Squares   F-statistic:                     497.3
Date:                Thu, 01 Dec 2022   Prob (F-statistic):               0.00
Time:                        15:34:53   Log-Likelihood:                -5634.7
No. Observations:                2049   AIC:                         1.132e+04
Df Residuals:                    2026   BIC:                         1.144e+04
Df Model:                          22
Covariance Type:            nonrobust
================================================================================
====================
                                     coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
--------------------
const                            -19.5814     39.815     -0.492      0.623
-97.665      58.502
Year                               0.0377      0.020      1.898      0.058
-0.001       0.077
Adult Mortality                   -0.0163      0.001    -17.884      0.000
-0.018      -0.014
Alcohol                           -0.0763      0.034     -2.276      0.023
-0.142      -0.011
Percentage expenditure             0.0003   4.97e-05      6.107      0.000
0.000       0.000
Hepatitis B                       -0.0160      0.004     -3.727      0.000
-0.024      -0.008
Measles                        -1.569e-05   9.36e-06     -1.676      0.094
-3.4e-05    2.67e-06
BMI                                0.0338      0.006      5.718      0.000
0.022       0.045
Under-five deaths                 -0.0020      0.001     -2.537      0.011
-0.004      -0.000
Polio                              0.0346      0.005      6.963      0.000
0.025       0.044
Total expenditure                 -0.0107      0.039     -0.272      0.786
-0.088       0.066
Diphtheria                         0.0341      0.005      6.603      0.000
0.024       0.044
HIV/AIDS                          -0.3807      0.021    -18.501      0.000
-0.421      -0.340
Population                      1.563e-09   2.32e-09      0.673      0.501   -
2.99e-09    6.12e-09
Thinness 5-9 years                -0.0827      0.029     -2.825      0.005
-0.140      -0.025
Income composition of resources    4.4975      0.725      6.205      0.000
3.076       5.919
Schooling                          0.6216      0.048     12.830      0.000
0.527       0.717
Status_Developing                 -2.6187      0.355     -7.369      0.000
-3.316      -1.922
Continent_Asia                     4.7377      0.282     16.821      0.000
4.185       5.290
Continent_Europe                   4.3452      0.412     10.535      0.000
3.536       5.154
Continent_North America            6.2346      0.364     17.113      0.000
5.520       6.949
Continent_Oceania                  2.7528      0.457      6.019      0.000
```

```
1.856       3.650
Continent_South America            4.3656      0.444      9.834      0.000
3.495       5.236
==============================================================================
Omnibus:                       79.888   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              213.118
Skew:                          -0.140   Prob(JB):                     5.27e-47
Kurtosis:                       4.555   Cond. No.                     2.07e+10
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.07e+10. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the Regression Results:

1. **Adjusted. R-squared**: It reflects the fit of the model.
   - Adjusted R-squared values generally range from 0 to 1, where a higher value generally indicates a better fit, assuming certain conditions are met.
   - In our case, the value for adj. R-squared is **0.842**, which is good!
2. *const* **coefficient**: It is the Y-intercept.
   - It means that if all the predictor variable coefficients are zero, then the expected output (i.e., Y) would be equal to the *const* coefficient.
   - In our case, the value for *const* coefficient is **-19.5814**
3. **Coefficient of a predictor variable**: It represents the change in the output Y due to a change in the predictor variable (everything else held constant).
   - In our case, the coefficient of `Adult Mortality` is **-0.0163**.
4. **std err**: It reflects the level of accuracy of the coefficients.
   - The lower it is, the higher is the level of accuracy.
5. **P>|t|**: It is p-value.
   - For each independent feature, there is a null hypothesis and an alternate hypothesis. Here β_i the coefficient of the i-th independent variable.
     - Null Hypothesis : Independent feature is not significant ($\beta_i = 0$)
     - Alternate Hypothesis : Independent feature is that it is significant ($\beta_i != 0$)
   - (P>|t|) gives the p-value for each independent feature to check that null hypothesis. We are considering 0.05 (5%) as significance level.
     - A p-value of less than 0.05 is considered to be statistically significant.
6. **Confidence Interval**: It represents the range in which our coefficients are likely to fall (with a likelihood of 95%).

Observations

- We can see that adj. R-squared has dropped from 0.845 to 0.842, which shows that the dropped columns did not have much effect on the model.
- As there is no multicollinearity, we can look at the p-values of predictor variables to check their significance.

Dealing with high p-value variables

- `const`, `year`, `Measles`, `Total expenditure`, and `Population` have p-value > 0.05. So, they are not significant and we'll drop them all except the constant.
    - In general, it is preferred not to remove the constant term from the linear regression as it ensures that the residuals are zero mean and the other regression parameters will be biased if it is removed, even if the constant is statistically insignificant..
- But sometimes p-values change after dropping a variable. So, we'll not drop all variables at once.
- Instead, we will do the following:
    - Build a model, check the p-values of the variables, and drop the column with the highest p-value.
    - Create a new model without the dropped feature, check the p-values of the variables, and drop the column with the highest p-value.
    - Repeat the above two steps till there are no columns with p-value > 0.05.

The above process can also be done manually by picking one variable at a time that has a high p-value, dropping it, and building a model again. But that might be a little tedious and using a loop will be more efficient.

In [87]:
```
# initial list of columns
cols = x_train4.columns.tolist()

# setting an initial max p-value
max_p_value = 1

while len(cols) > 0:
    # defining the train set
    x_train_aux = x_train4[cols]

    # fitting the model
    model = sm.OLS(y_train, x_train_aux).fit()

    # getting the p-values and the maximum p-value
    p_values = model.pvalues
    max_p_value = max(p_values)

    # name of the variable with maximum p-value
    feature_with_p_max = p_values.idxmax()

    if max_p_value > 0.05:
        cols.remove(feature_with_p_max)
    else:
        break

selected_features = cols
print(selected_features)
```

```
['Year', 'Adult Mortality', 'Alcohol', 'Percentage expenditure', 'Hepatitis B',
'BMI', 'Under-five deaths', 'Polio', 'Diphtheria', 'HIV/AIDS', 'Thinness 5-9 ye
ars', 'Income composition of resources', 'Schooling', 'Status_Developing', 'Con
tinent_Asia', 'Continent_Europe', 'Continent_North America', 'Continent_Oceani
a', 'Continent_South America']
```

In [88]:
```
x_train5 = x_train4[["const"] + selected_features]
x_test5 = x_test4[["const"] + selected_features]
```

In [89]:
```
olsmod2 = sm.OLS(y_train, x_train5).fit()
print(olsmod2.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:        Life expectancy   R-squared:                       0.843
Model:                            OLS   Adj. R-squared:                  0.842
Method:                 Least Squares   F-statistic:                     575.6
Date:                Thu, 01 Dec 2022   Prob (F-statistic):               0.00
Time:                        15:35:22   Log-Likelihood:                -5636.5
No. Observations:                2049   AIC:                         1.131e+04
Df Residuals:                    2029   BIC:                         1.143e+04
Df Model:                          19                                         
Covariance Type:            nonrobust                                         
===============================================================================
===================
                                      coef    std err          t      P>|t|      
[0.025      0.975]
-------------------------------------------------------------------------------
-------------------
const                             -23.1832     39.635     -0.585      0.559     
-100.913      54.547
Year                                0.0395      0.020      1.994      0.046     
0.001       0.078
Adult Mortality                    -0.0162      0.001    -17.819      0.000     
-0.018      -0.014
Alcohol                            -0.0801      0.033     -2.395      0.017     
-0.146      -0.015
Percentage expenditure              0.0003   4.96e-05      6.128      0.000     
0.000       0.000
Hepatitis B                        -0.0163      0.004     -3.821      0.000     
-0.025      -0.008
BMI                                 0.0346      0.006      5.860      0.000     
0.023       0.046
Under-five deaths                  -0.0023      0.001     -3.794      0.000     
-0.004      -0.001
Polio                               0.0346      0.005      6.960      0.000     
0.025       0.044
Diphtheria                          0.0344      0.005      6.688      0.000     
0.024       0.045
HIV/AIDS                           -0.3813      0.020    -18.604      0.000     
-0.422      -0.341
Thinness 5-9 years                 -0.0777      0.029     -2.669      0.008     
-0.135      -0.021
Income composition of resources     4.5468      0.721      6.302      0.000     
3.132       5.962
Schooling                           0.6184      0.048     12.789      0.000     
0.524       0.713
Status_Developing                  -2.6234      0.353     -7.442      0.000     
-3.315      -1.932
Continent_Asia                      4.7406      0.281     16.862      0.000     
4.189       5.292
Continent_Europe                    4.3902      0.411     10.694      0.000     
3.585       5.195
Continent_North America             6.2753      0.360     17.417      0.000     
5.569       6.982
Continent_Oceania                   2.7757      0.456      6.089      0.000     
1.882       3.670
Continent_South America             4.4261      0.440     10.062      0.000     
3.563       5.289
==============================================================================
Omnibus:                       80.001   Durbin-Watson:                   2.014
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              214.140
```

```
Skew:                          -0.138   Prob(JB):                     3.16e-47
Kurtosis:                       4.559   Cond. No.                     1.14e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.14e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Now no feature (other than the constant) has p-value greater than 0.05, so we'll consider the features in *x_train5* as the final set of predictor variables and *olsmod2* as final model.

Observations

- Now adjusted R-squared is 0.842, i.e., our model is able to explain ~84% of the variance. This shows that the model is good.
- The adjusted R-squared in *olsmod0* (where we considered all the variables) was 0.845. This shows that the variables we dropped were not affecting the model much.

Now we'll check the rest of the assumptions on *olsmod2*.

2. **Linearity of variables**
3. **Independence of error terms**
4. **Normality of error terms**
5. **No Heteroscedasticity**

TEST FOR LINEARITY AND INDEPENDENCE

Why the test?

- Linearity describes a straight-line relationship between two variables, predictor variables must have a linear relation with the dependent variable.
- The independence of the error terms (or residuals) is important. If the residuals are not independent, then the confidence intervals of the coefficient estimates will be narrower and make us incorrectly conclude a parameter to be statistically significant.

How to check linearity and independence?

- Make a plot of fitted values vs residuals.
- If they don't follow any pattern, then we say the model is linear and residuals are independent.
- Otherwise, the model is showing signs of non-linearity and residuals are not independent.

How to fix if this assumption is not followed?

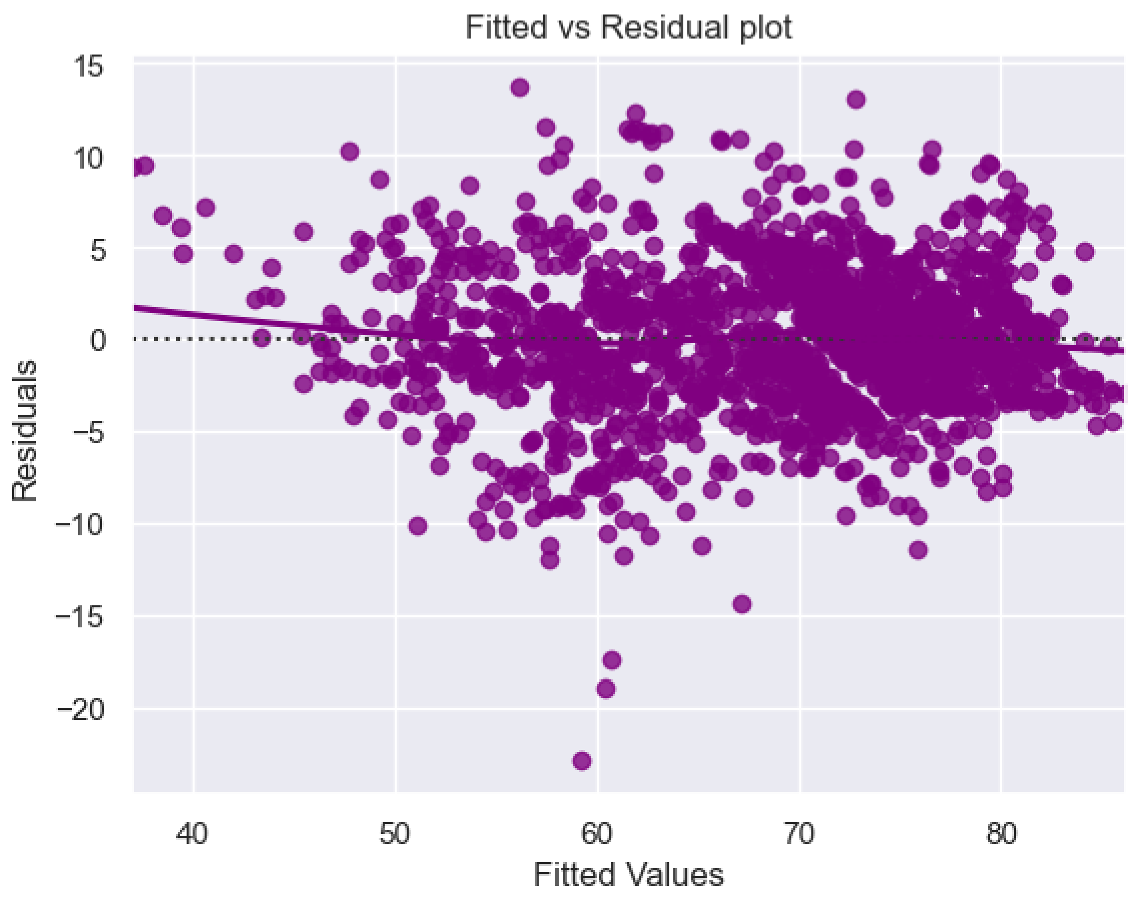
- We can try to transform the variables and make the relationships linear.

In [90]:
```
# let us create a dataframe with actual, fitted and residual values
df_pred = pd.DataFrame()

df_pred["Actual Values"] = y_train  # actual values
df_pred["Fitted Values"] = olsmod2.fittedvalues  # predicted values
df_pred["Residuals"] = olsmod2.resid  # residuals

df_pred.head()
```

Out[90]:

	Actual Values	Fitted Values	Residuals
608	64.7	58.396287	6.303713
348	46.4	46.844307	-0.444307
1410	71.1	72.897591	-1.797591
1730	63.2	68.012416	-4.812416
612	62.9	61.834162	1.065838

In [91]:
```
# let's plot the fitted values vs residuals

sns.residplot(
    data=df_pred, x="Fitted Values", y="Residuals", color="purple", lowess=True
)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Fitted vs Residual plot")
plt.show()
```



- The scatter plot shows the distribution of residuals (errors) vs fitted values (predicted values).
- If there exist any pattern in this plot, we consider it as signs of non-linearity in the data and a pattern means that the model doesn't capture non-linear effects.
- **We see no pattern in the plot above. Hence, the assumptions of linearity and independence are satisfied.**

TEST FOR NORMALITY

Why the test?

- Error terms, or residuals, should be normally distributed. If the error terms are not normally distributed, confidence intervals of the coefficient estimates may become too wide or narrow. Once confidence interval becomes unstable, it leads to difficulty in estimating coefficients based on minimization of least squares. Non-normality suggests that there are a few unusual data points that must be studied closely to make a better model.

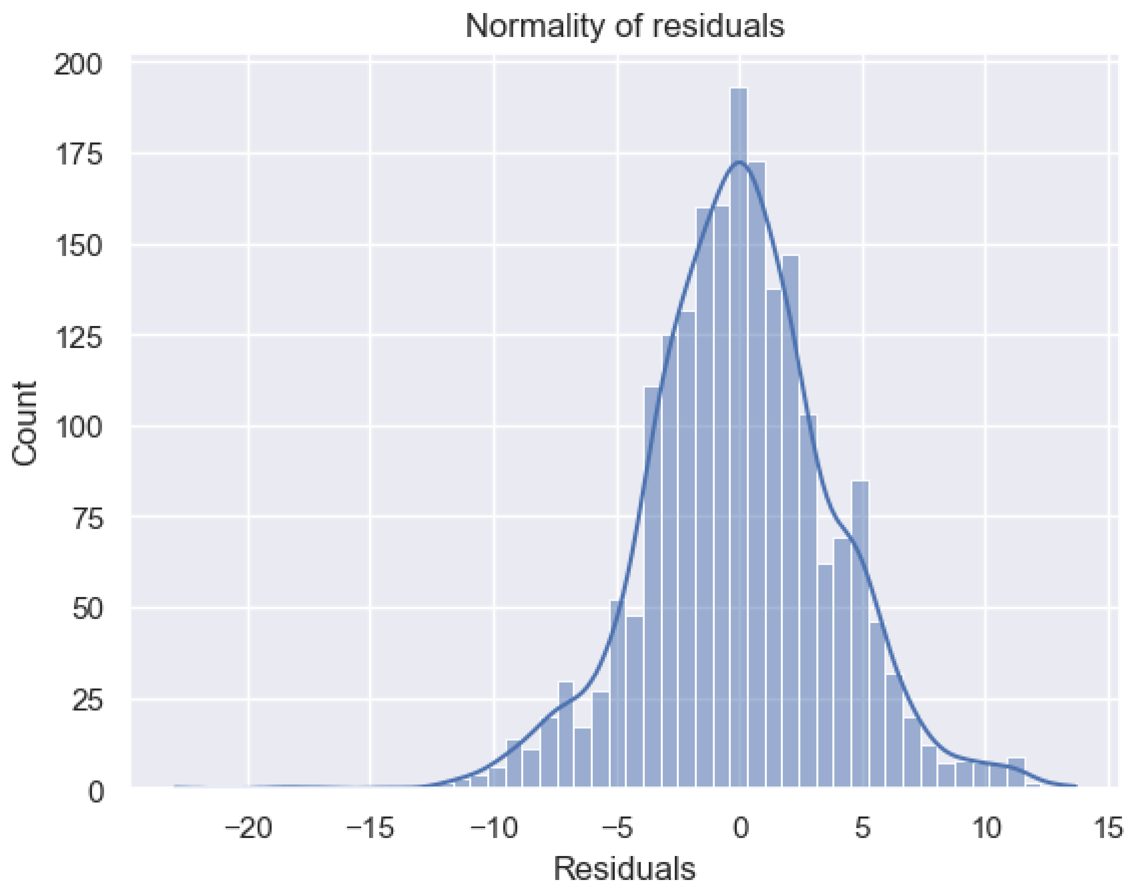
How to check normality?

- The shape of the histogram of residuals can give an initial idea about the normality.
- It can also be checked via a Q-Q plot of residuals. If the residuals follow a normal distribution, they will make a straight line plot, otherwise not.
- Other tests to check for normality includes the Shapiro-Wilk test.
  - Null hypothesis: Residuals are normally distributed
  - Alternate hypothesis: Residuals are not normally distributed

How to fix if this assumption is not followed?

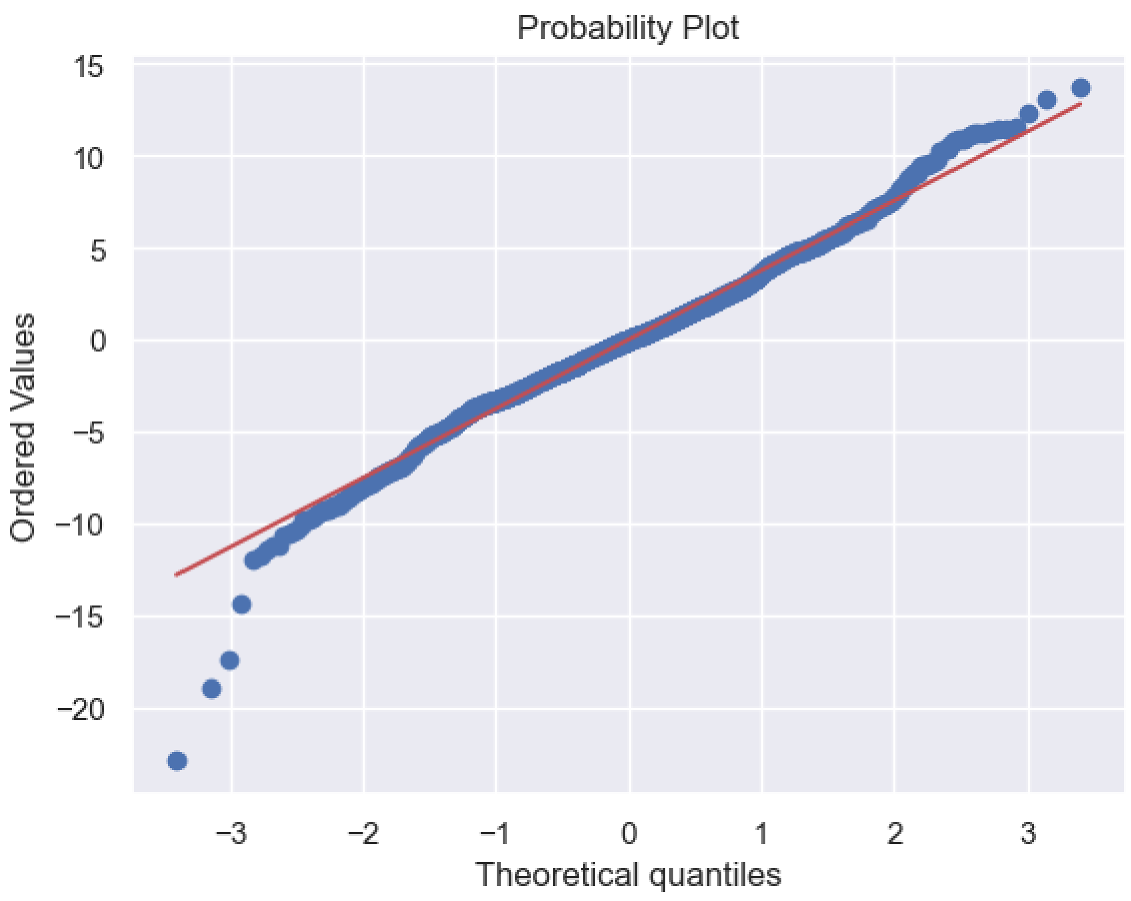
- We can apply transformations like log, exponential, arcsinh, etc. as per our data.

In [92]:
```
sns.histplot(data=df_pred, x="Residuals", kde=True)
plt.title("Normality of residuals")
plt.show()
```



- The histogram of residuals does have a bell shape.
- Let's check the Q-Q plot.

In [93]:
```
import pylab
import scipy.stats as stats

stats.probplot(df_pred["Residuals"], dist="norm", plot=pylab)
plt.show()
```



- The residuals more or less follow a straight line except for the tails.
- Let's check the results of the Shapiro-Wilk test.

In [94]:
```
stats.shapiro(df_pred["Residuals"])
```

Out[94]:
```
ShapiroResult(statistic=0.9869110584259033, pvalue=9.742823952121893e-13)
```

- Since p-value < 0.05, the residuals are not normal as per the Shapiro-Wilk test.
- Strictly speaking, the residuals are not normal.
- However, as an approximation, we can accept this distribution as close to being normal.
- **So, the assumption is satisfied.**

TEST FOR HOMOSCEDASTICITY

- **Homoscedascity**: If the variance of the residuals is symmetrically distributed across the regression line, then the data is said to be homoscedastic.

- **Heteroscedascity**: If the variance is unequal for the residuals across the regression line, then the data is said to be heteroscedastic.

Why the test?

- The presence of non-constant variance in the error terms results in heteroscedasticity. Generally, non-constant variance arises in presence of outliers.

How to check for homoscedasticity?

- The residual vs fitted values plot can be looked at to check for homoscedasticity. In the case of heteroscedasticity, the residuals can form an arrow shape or any other non-symmetrical shape.
- The goldfeldquandt test can also be used. If we get a p-value > 0.05 we can say that the residuals are homoscedastic. Otherwise, they are heteroscedastic.
    - Null hypothesis: Residuals are homoscedastic
    - Alternate hypothesis: Residuals have heteroscedasticity

How to fix if this assumption is not followed?

- Heteroscedasticity can be fixed by adding other important features or making transformations.

In [95]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], x_train5)
lzip(name, test)
```

Out[95]:
```
[('F statistic', 0.9830406677988949), ('p-value', 0.6067961354027434)]
```

Since p-value > 0.05, we can say that the residuals are homoscedastic. So, this assumption is satisfied.

Now that we have checked all the assumptions of linear regression and they are satisfied, let's go ahead with prediction.

In [96]:
```
# predictions on the test set
pred = olsmod2.predict(x_test5)

df_pred_test = pd.DataFrame({"Actual": y_test, "Predicted": pred})
df_pred_test.sample(10, random_state=1)
```

Out[96]:

	Actual	Predicted
45	71.6	66.185783
1294	79.9	79.784195
187	75.2	75.787198
661	78.8	78.745847
2788	52.2	55.639856
151	73.0	72.287647
2845	71.4	67.126506
1639	83.0	78.023254
393	72.2	73.905880
1607	76.7	71.833115

- We can observe here that our model has returned pretty good prediction results, and the actual and predicted values are comparable.

In [97]:
```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmod2_train_perf = model_performance_regression(olsmod2, x_train5, y_train)
olsmod2_train_perf
```

```
Training Performance
```

Out[97]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.788223	2.869585	0.843498	0.841954	4.369148

In [98]:
```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmod2_test_perf = model_performance_regression(olsmod2, x_test5, y_test)
olsmod2_test_perf
```

```
Test Performance
```

Out[98]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.770886	2.854912	0.838908	0.835153	4.333436

- The model is able to explain ~84% of the variation in the data, which is very good.
- The train and test RMSE and MAE are low and comparable. So, our model is not suffering from overfitting.
- The MAPE on the test set suggests we can predict within 4.3% of the life expectancy.
- Hence, we can conclude the model *olsmod2* is good for prediction as well as inference purposes.

In [99]:
```
from sklearn.linear_model import LinearRegression
linearregression = LinearRegression()
linearregression.fit(x_train5, y_train)

print("Intercept of the linear equation:", linearregression.intercept_)
print("\nCOefficients of the equation are:", linearregression.coef_)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

pred = linearregression.predict(x_test5)
```

```
Intercept of the linear equation: -23.183230820132863

COefficients of the equation are: [ 0.00000000e+00  3.94781648e-02 -1.61684228e
-02 -8.00932387e-02
  3.04140832e-04 -1.63320129e-02  3.45821648e-02 -2.33205519e-03
  3.45661456e-02  3.44268946e-02 -3.81323894e-01 -7.76853017e-02
  4.54681146e+00  6.18422003e-01 -2.62341318e+00  4.74056761e+00
  4.39016884e+00  6.27533918e+00  2.77573056e+00  4.42612657e+00]
```

In [101…

```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
linreg_train_perf = model_performance_regression(linearregression, x_train5, y_t
linreg_train_perf
```

```
Training Performance
```

Out[101]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.788223	2.869585	0.843498	0.841954	4.369148

In [103…

```
# checking model performance on testing set (seen 70% data)
print("Testing Performance\n")
linreg_test_perf = model_performance_regression(linearregression, x_test5, y_tes
linreg_test_perf
```

```
Testing Performance
```

Out[103]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	3.770886	2.854912	0.838908	0.835153	4.333436

- The model is able to explain ~84% of the variation in the data, which is very good.
- The train and test RMSE and MAE are low and comparable. So, our model is not suffering from overfitting.
- The MAPE on the test set suggests we can predict within 4.3% of the life expectancy.
- Hence, we can conclude the model *Linrear Regression* is good for prediction as well as inference purposes.

In [120…

```
cust1 = x_test5.iloc[100].to_dict()
cust1
```

Out[120]:

```
{'const': 1.0,
 'Year': 2011.0,
 'Adult Mortality': 464.0,
 'Alcohol': 6.0,
 'Percentage expenditure': 63.75053034,
 'Hepatitis B': 94.0,
 'BMI': 29.9,
 'Under-five deaths': 42.0,
 'Polio': 93.0,
 'Diphtheria': 93.0,
 'HIV/AIDS': 13.3,
 'Thinness 5-9 years': 6.7,
 'Income composition of resources': 0.452,
 'Schooling': 10.1,
 'Status_Developing': 1.0,
 'Continent_Asia': 0.0,
 'Continent_Europe': 0.0,
 'Continent_North America': 0.0,
 'Continent_Oceania': 0.0,
 'Continent_South America': 0.0}
```

In [129…

```
linearregression.predict(np.array(x_test5.iloc[1]).reshape(1,-1))[0]
```

```
c:\Users\pavanksu2009\.virtualenvs\system32-zwnXhztR\lib\site-packages\sklearn
\base.py:450: UserWarning: X does not have valid feature names, but LinearRegre
ssion was fitted with feature names
  warnings.warn(
```

Out[129]:

```
69.03474620553422
```

In [ ]: