In [ ]:

```
# !pip install -U kaleido
```

In [ ]:

```
# !pip install --upgrade "kaleido==0.1.*"
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: kaleido==0.1.* in /usr/local/lib/python3.7/dist-packages (0.1.0)
```

In [135]:

```
# importing all the necessary liraries/modules

import io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import json
import math
import kaleido
from sklearn.datasets import load_boston
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from yellowbrick.regressor import ResidualsPlot
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import scale
from sklearn.compose import TransformedTargetRegressor
from sklearn.preprocessing import QuantileTransformer
from sklearn.ensemble import GradientBoostingRegressor
import lightgbm
import xgboost
```

Importing the dataset:

In [2]:

```
# Importing the dataset onto python
df = pd.read_csv('Life Expectancy Data_HV22.csv')
```

In [3]:
```python
# Moving the Target column to the end

def moveColumn(df, column_name, position):
  column_to_move = df.pop(column_name)
  df.insert(position, column_name, column_to_move)

moveColumn(df, 'Life expectancy ', 21)
```

In [4]:
```python
df.head()
```

Out[4]:

	Country	Year	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling	Life expectancy
0	Afghanistan	2015	Developing	263.0	62	0.01	71.279624	65.0	1154	19.1	...	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3	0.479	10.1	65.0
1	Afghanistan	2014	Developing	271.0	64	0.01	73.523582	62.0	492	18.6	...	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5	0.476	10.0	59.9
2	Afghanistan	2013	Developing	268.0	66	0.01	73.219243	64.0	430	18.1	...	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7	0.470	9.9	59.9
3	Afghanistan	2012	Developing	272.0	69	0.01	78.184215	67.0	2787	17.6	...	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0	0.463	9.8	59.5
4	Afghanistan	2011	Developing	275.0	71	0.01	7.097109	68.0	3013	17.2	...	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2	0.454	9.5	59.2

5 rows × 22 columns

Basic Summary about df:

In [5]:
```python
# Basic statistical summary of the df
df.describe()
```

Out[5]:

	Year	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	co
count	2938.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.000000	2938.000000	2919.000000	2712.00000	2919.000000	2938.000000	2490.000000	2.286000e+03	2904.000000	2904.000000	27
mean	2007.518720	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.321247	42.035739	82.550188	5.93819	82.324084	1.742103	7483.158469	1.275338e+07	4.839704	4.870317	
std	4.613841	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.044034	160.445548	23.428046	2.49832	23.716912	5.077785	14270.169342	6.101210e+07	4.420195	4.508882	
min	2000.000000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000000	3.000000	0.37000	2.000000	0.100000	1.681350	3.400000e+01	0.100000	0.100000	
25%	2004.000000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.300000	0.000000	78.000000	4.26000	78.000000	0.100000	463.935626	1.957932e+05	1.600000	1.500000	
50%	2008.000000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	43.500000	4.000000	93.000000	5.75500	93.000000	0.100000	1766.947595	1.386542e+06	3.300000	3.300000	
75%	2012.000000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	56.200000	28.000000	97.000000	7.49250	97.000000	0.800000	5910.806335	7.420359e+06	7.200000	7.200000	
max	2015.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000000	99.000000	17.60000	99.000000	50.600000	119172.741800	1.293859e+09	27.700000	28.600000	

In [6]:
```
# data types of all the columns
df.dtypes
```

Out[6]:
```
Country                             object
Year                                 int64
Status                              object
Adult Mortality                    float64
infant deaths                        int64
Alcohol                            float64
percentage expenditure             float64
Hepatitis B                        float64
Measles                              int64
 BMI                               float64
under-five deaths                    int64
Polio                              float64
Total expenditure                  float64
Diphtheria                         float64
 HIV/AIDS                          float64
GDP                                float64
Population                         float64
 thinness  1-19 years              float64
 thinness 5-9 years                float64
Income composition of resources    float64
Schooling                          float64
Life expectancy                    float64
dtype: object
```

In [7]:
```
# Prnitin the basic idea about the df
print("Total number of NA values in df:", df.isna().sum().sum(), "\nTotal number of columns in df:", len(df.columns), "\nThere are", len(df['Country'].unique()), "countries in total.")
```

```
Total number of NA values in df: 2563
Total number of columns in df: 22
There are 193 countries in total.
```

Dealing with all the missing values:

In [8]:
```
# Calculating the number of NA values in each column
df.isna().sum()
```

Out[8]:
```
Country                              0
Year                                 0
Status                               0
Adult Mortality                     10
infant deaths                        0
Alcohol                            194
percentage expenditure               0
Hepatitis B                        553
Measles                              0
 BMI                                34
under-five deaths                    0
Polio                               19
Total expenditure                  226
Diphtheria                          19
 HIV/AIDS                            0
GDP                                448
Population                         652
 thinness  1-19 years               34
 thinness 5-9 years                 34
Income composition of resources    167
Schooling                          163
Life expectancy                     10
dtype: int64
```

Country

In [9]:
```
# Eliminating all the countries with only 1 instance
countries = df["Country"].value_counts()
oneCountry = list(countries[countries == 1].index)
oneCountry
```

Out[9]:
```
['Tuvalu',
 'Cook Islands',
 'Marshall Islands',
 'Monaco',
 'Palau',
 'Niue',
 'San Marino',
 'Nauru',
 'Saint Kitts and Nevis',
 'Dominica']
```

In [10]:
```
for element in df["Country"]:
  if element in oneCountry:
    indices = df[df["Country"]==element].index
    df.drop(indices,axis=0, inplace=True)
```

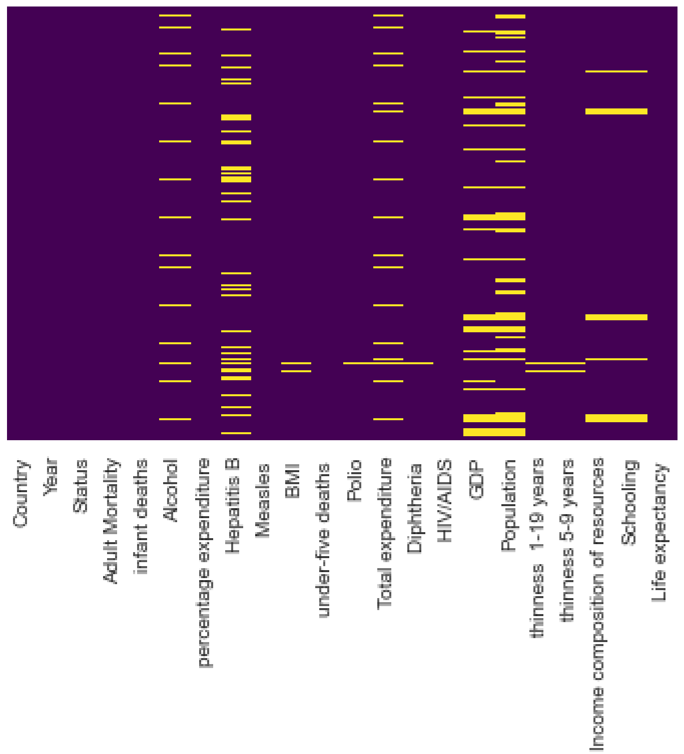
In [11]:
```
# resetting the index of the df
df.reset_index(drop=True, inplace=True)
```

Rest of the columns

In [12]:

```
# Plotting a correlation heatmap for all the NaN values
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap="viridis")
```

Out[12]: <AxesSubplot:>



In [13]:

```
# Filling in all the NaN values through the 'interpolate' method

df.reset_index(inplace=True)
df.groupby('Country').apply(lambda group: group.interpolate(method= 'linear'))
imputed_data = []
for year in list(df.Year.unique()):
    year_data = df[df.Year == year].copy()
    for col in list(year_data.columns)[4:]:
        year_data[col] = year_data[col].fillna(year_data[col].dropna().median()).copy()
    imputed_data.append(year_data)
df = pd.concat(imputed_data).copy()
```

```
<ipython-input-13-e45010ce39de>:4: FutureWarning: Not prepending group keys to the result index of transform-like apply. In the future, the group keys will be included in the index, regardless of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df.groupby('Country').apply(lambda group: group.interpolate(method= 'linear'))
```

In [14]:

```
print("The number of NaN values in the df:", df.isna().sum().sum())
```

```
The number of NaN values in the df: 0
```

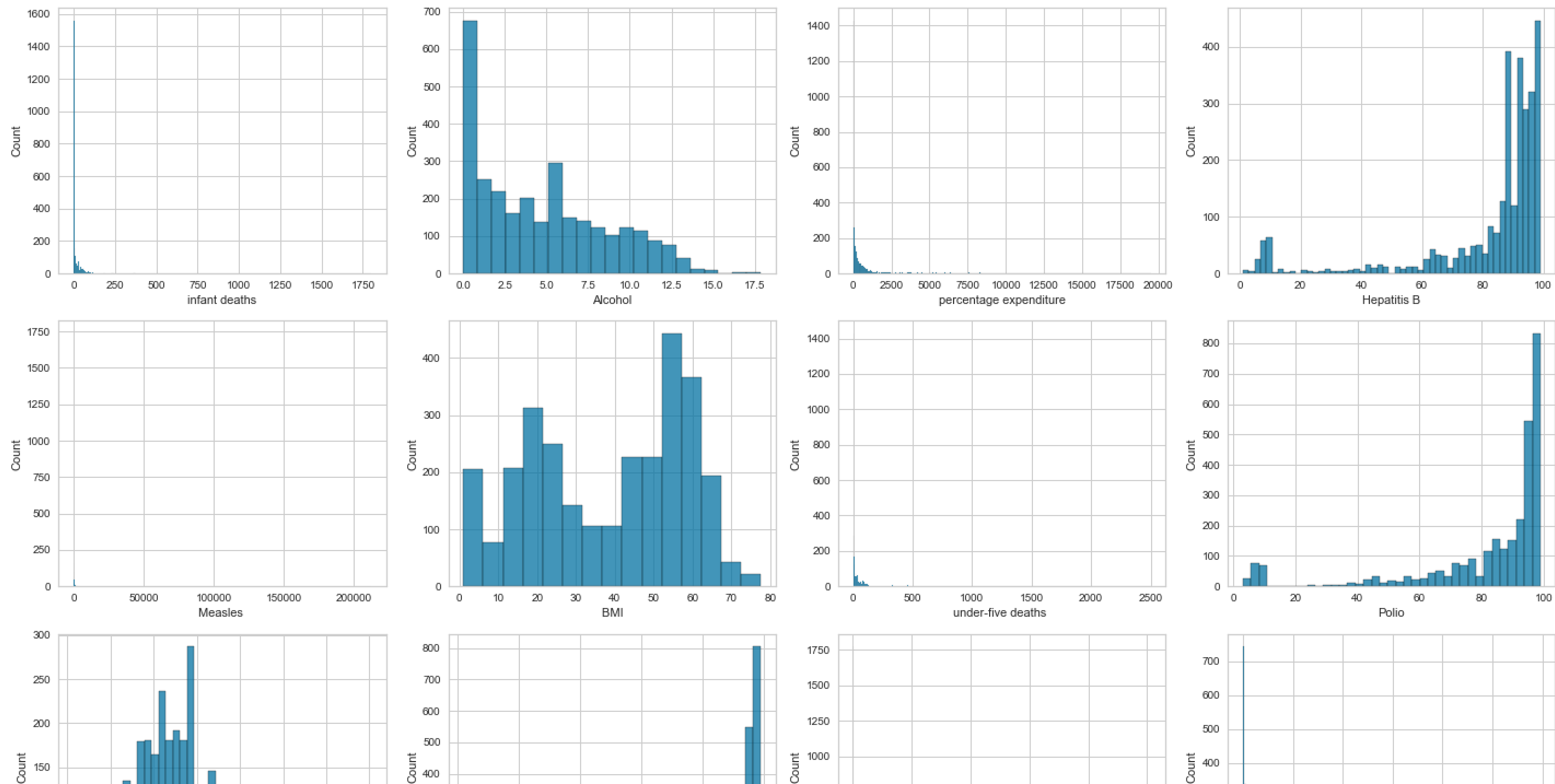
EDA:

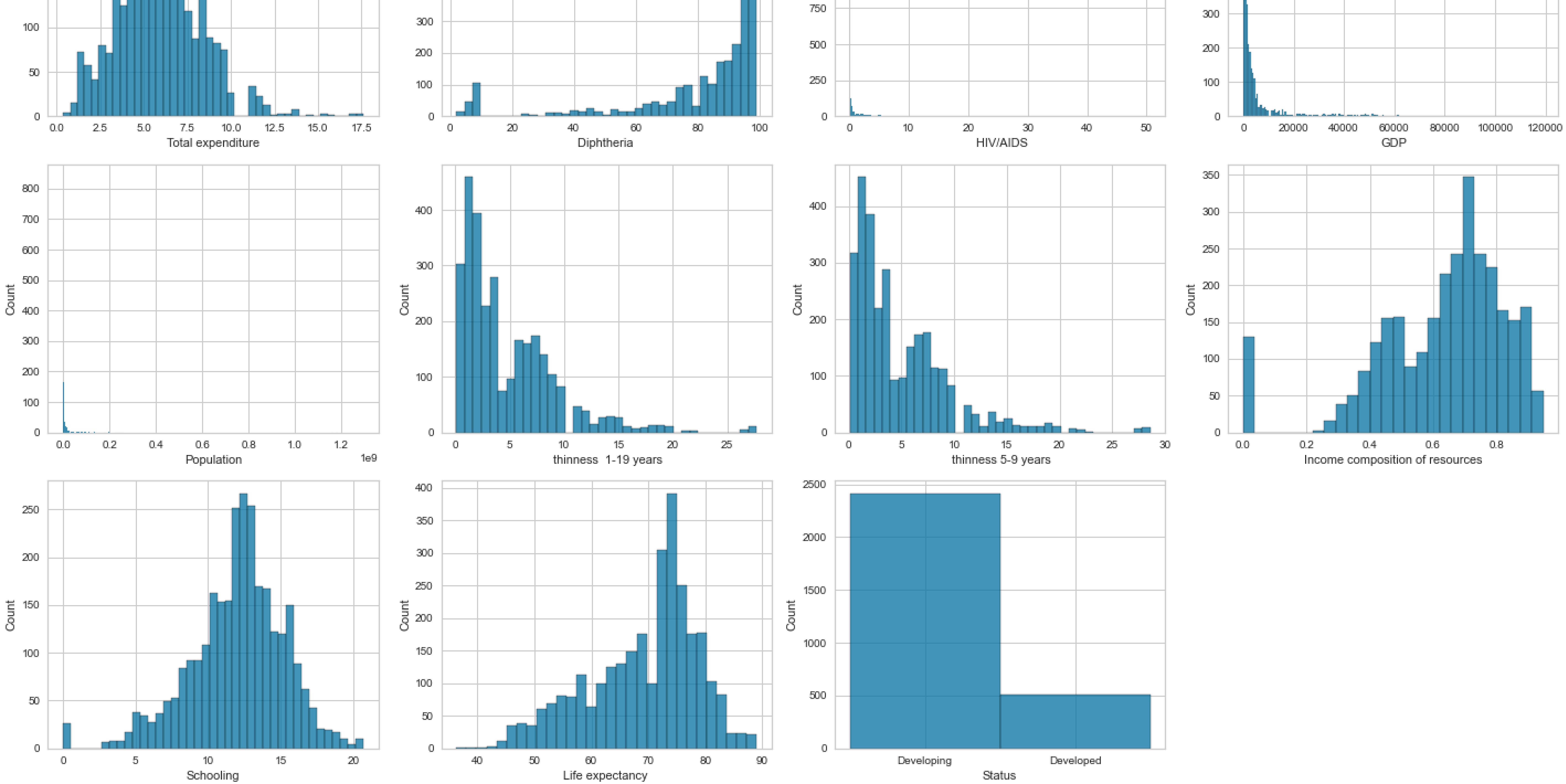
1. Basic Plots

1.1) Counting Plots

In [15]:

```python
col_list = ['Year', 'Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure', 'Diphtheria
        ' thinness  1-19 years', ' thinness 5-9 years', 'Income composition of resources', 'Schooling', 'Life expectancy ', 'Status']

plt.figure(figsize=(20,20))
column_list = col_list[2:]
plt_num = 1

for i in column_list:
    if plt_num<=18:
        plt.subplot(5, 4, plt_num)
        sns.histplot(df[i])
        plt_num = plt_num+1
    else:
        plt.subplot(5, 4, plt_num)
        sns.histplot(df[i])
        plt_num = plt_num+1
plt.tight_layout()
```

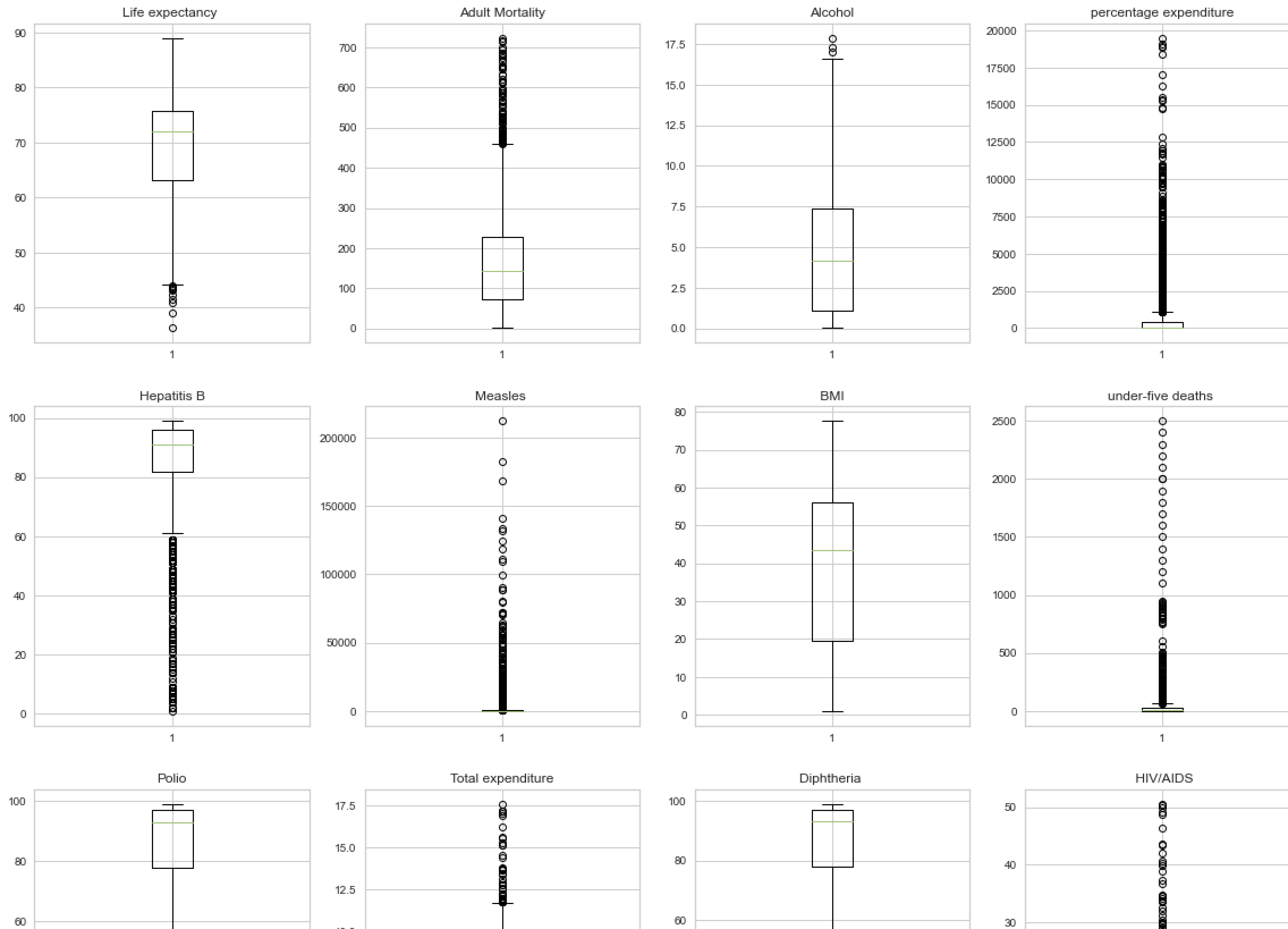


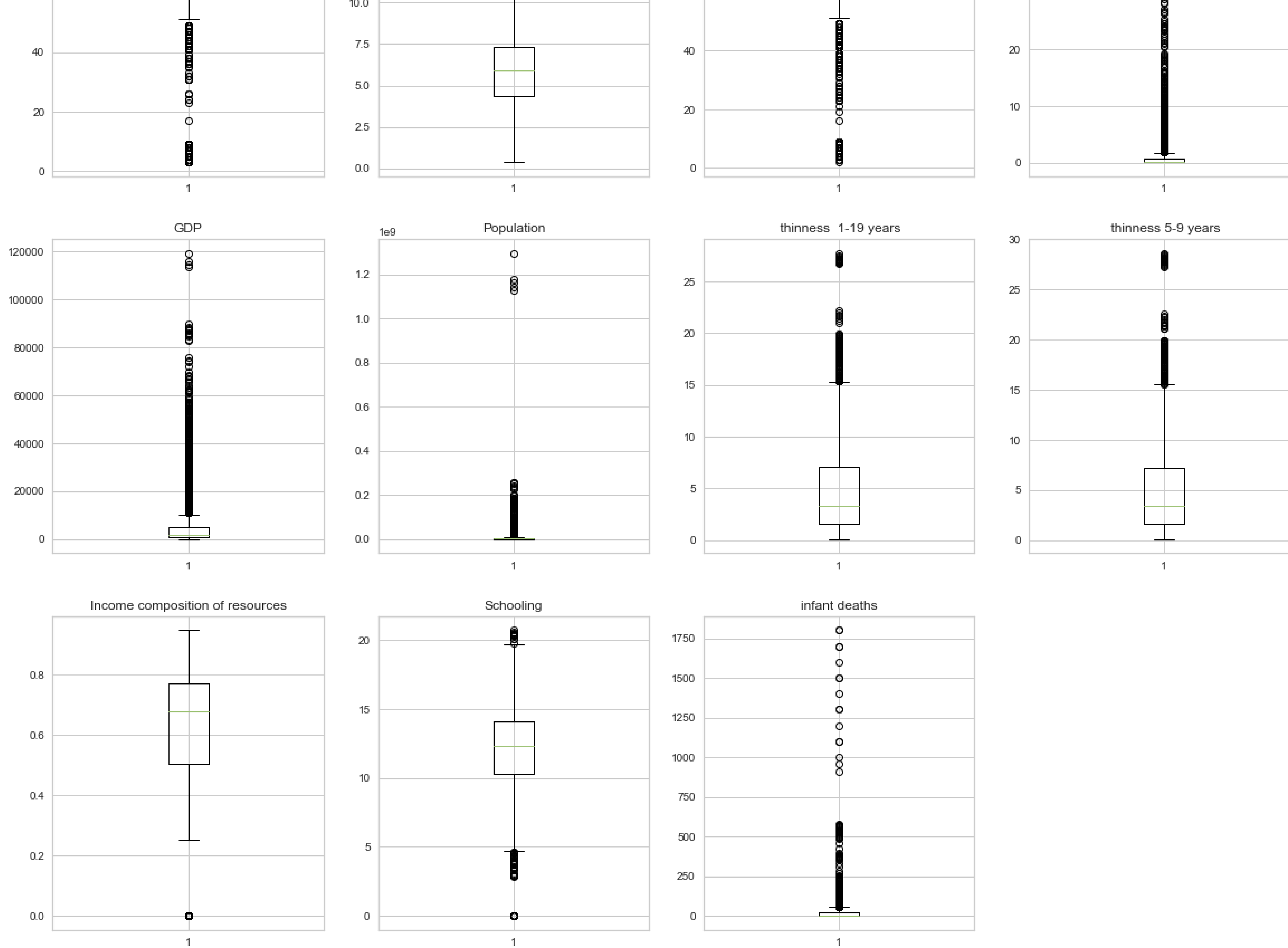


1.2) Box Plots for Outliers

In [16]:

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

plt.figure(figsize=(20,30))

for variable,i in col_dict.items():
                     plt.subplot(5,4,i)
                     plt.boxplot(df[variable],whis=1.5)
                     plt.title(variable)

plt.show()
```





2. Automated EDA

In [17]:
```
# pip install dataprep
```

In [18]:

```
'''
from dataprep.eda import create_report
report = create_report(df, title='My Report')
report
'''
```

Out[18]: "\nfrom dataprep.eda import create_report\nreport = create_report(df, title='My Report')\nreport\n"

Overview

Dataset Statistics

Number of Variables	205
Number of Rows	2928
Missing Cells	0
Missing Cells (%)	0.0%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	4.6 MB
Average Row Size in Memory	1.6 KB
Variable Types	Numerical: 20 Categorical: 185

Dataset Insights

`Polio` and `Diphtheria` have similar distributions	Similar Distribution
`thinness 1-19 years` and `thinness 5-9 years` have similar distributions	Similar Distribution
`infant deaths` is skewed	Skewed
`Alcohol` is skewed	Skewed
`percentage expenditure` is skewed	Skewed
`Hepatitis B` is skewed	Skewed
`Measles` is skewed	Skewed
`under-five deaths` is skewed	Skewed
`Polio` is skewed	Skewed
`Diphtheria` is skewed	Skewed

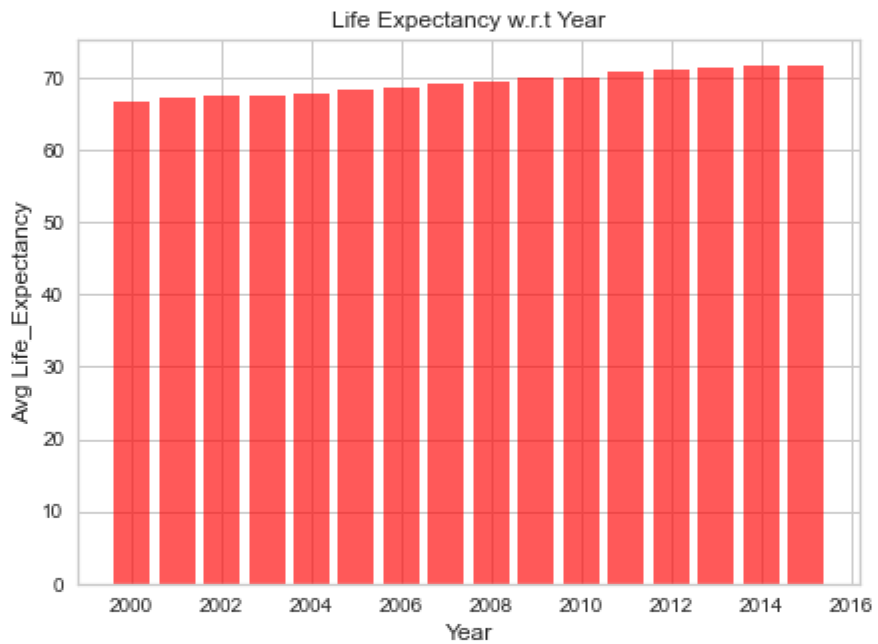
1 2 3 4 5 6 7

3. Manual Plotting

3.1) Life Expectancy vs Year

In [19]:

```
plt.figure(figsize=(7,5))
plt.bar(df.groupby('Year')['Year'].count().index, df.groupby('Year')['Life expectancy '].mean(),color='red',alpha=0.65)
plt.xlabel("Year",fontsize=12)
plt.ylabel("Avg Life_Expectancy",fontsize=12)
plt.title("Life Expectancy w.r.t Year")
plt.show()
```



3.2) Life Expectancy vs Hepatitis B

The Developed countries' regression line is slightly tilted downwards stating that the Developed countries' status on getting the vaccine for Hepatitis B is gradually decreasing whereas the contrary can be noticed in the case of Developed countries.

Feature Engineering:

Label Encoding the categorical variable

In [20]:

```python
# One Hot Encoding the Status column
# One Hot Encoding basically splits the unique values of a column to different columns

one_hot_encoder_status = OneHotEncoder()
status_reshaped = np.array(df['Status']).reshape(-1, 1)
status_values = one_hot_encoder_status.fit_transform(status_reshaped)

status_col = df['Status'].unique()
status_df = pd.DataFrame(status_values.toarray(), columns = status_col)

df = pd.concat([df, status_df], axis = 1)
df = df.drop(['Status'], axis = 1)
```

In [21]:

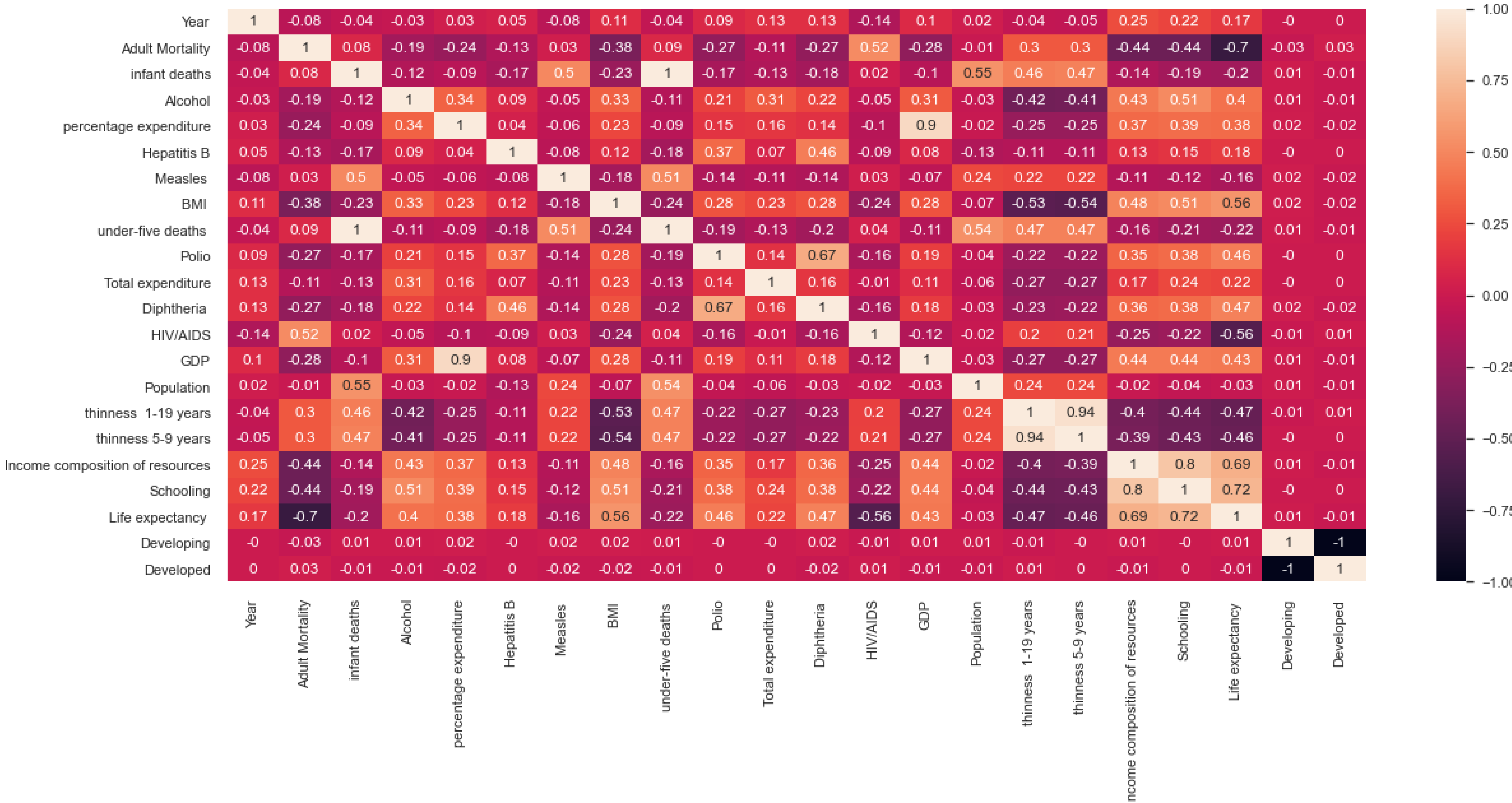
```python
df.drop(['index'], axis = 1, inplace = True)
```

In [22]:

```python
one_hot_encoder_country = OneHotEncoder()
country_reshaped = np.array(df['Country']).reshape(-1, 1)
country_values = one_hot_encoder_country.fit_transform(country_reshaped)

country_col = df['Country'].unique()
country_df = pd.DataFrame(country_values.toarray(), columns = country_col)

df = pd.concat([df, country_df], axis = 1)
df = df.drop(['Country'], axis = 1)
```

In [23]:

```
# Selecting columns with moderate to high correlation values
sns.set(rc={'figure.figsize':(20, 8.27)})
corr_map = sns.heatmap(df[['Year', 'Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditur
       ' HIV/AIDS', 'GDP', 'Population', ' thinness  1-19 years', ' thinness 5-9 years', 'Income composition of resources', 'Schooling', 'Life expectancy ', 'Developing', 'Developed']].corr().
```



Outlier Analysis:

In [24]:
```
def outlierAnalysis(df, col_name):

  df_score = pd.DataFrame(zscore(df[col_name]))

  score_index = []
  for i in range(len(df_score)):
    if (df[col_name][i]>3) or (df[col_name][i]<-3):
      score_index.append(i)
    else:
      pass

  return score_index
```

In [25]:
```
# outlierAnalysis(df, 'Polio')
```

Each column seems to have great huge amount of rows with absolute value so, we avoid using this methd to inrease the efficiency of the model.

Training a Linear Regression model

In [145]:
```
# Splitting the data into Features and Targets

X = df.drop('Life expectancy ', axis = 1) # X --> Features
X = X.apply(pd.to_numeric, errors='coerce')
y = df['Life expectancy '] # y --> Target Variable
y = y.apply(pd.to_numeric, errors='coerce')
```

In [151]:
```
# Splitting the data into training data and testing data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 500)
linear_regressor = LinearRegression()
```

In [152]:

```python
# Fitting the Data
linear_regressor.fit(X_train, y_train)

# Predicting the Target Variable
y_pred = linear_regressor.predict(X_test)
```

Regression Plot

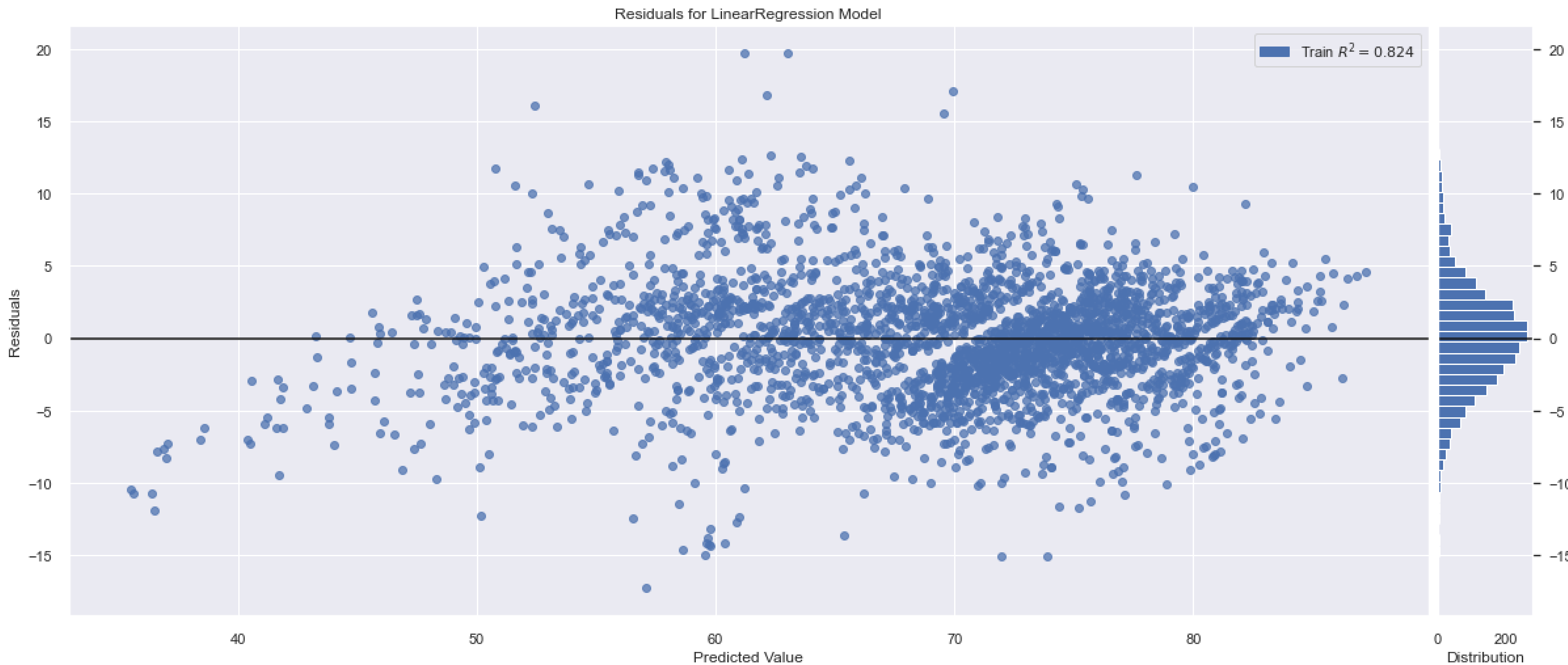
In [153]:

```python
# Plotting the true and predicted value
fig = plt.figure(figsize =(20, 10))
sns.regplot(y_test, y_pred)
plt.xlabel("LE Test")
plt.ylabel("LE Predicted")
plt.title("True Value vs Predicted Value")
plt.show()
```

```
C:\Users\joanj\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

In [154]:
```
# Predicted Valule vs Residuals
visualizer = ResidualsPlot(estimator = linear_regressor)
visualizer.fit(X, y)
visualizer.poof()
```

```
C:\Users\joanj\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```



Out[154]:
```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>
```

Accuracy Count

In [155]:
```
# Chcking the accuracy of the model
print("The accuracy of the model is", linear_regressor.score(X_test, y_test))
```

```
The accuracy of the model is 0.8116355298655129
```

In [156]:
```
print("The r2 score of the model is", r2_score(y_test, y_pred))
```

```
The r2 score of the model is 0.8116355298655129
```

Increasing the accuracy

In [157]:
```
model = DecisionTreeRegressor()

regr_trans = TransformedTargetRegressor(regressor = model, transformer=QuantileTransformer(output_distribution='normal'))
regr_trans.fit(X_train, y_train)
yhat = regr_trans.predict(X_test)
```

In [160]:
```
print("The accuracy of the model is", regr_trans.score(X_test, y_test))
```

```
The accuracy of the model is 0.9073821902440397
```

In [161]:
```
print("The r2 score of the model is", r2_score(y_test, yhat))
```

```
The r2 score of the model is 0.9073821902440397
```

Evaluation Metrics

In [162]:
```
# Evaluation Metrics
print("The Absolute Mean Error of the model is", mean_absolute_error(y_test, yhat))
```

```
The Absolute Mean Error of the model is 1.7109215017064847
```

In [163]:
```
print("The Mean Squared Error of the model is", mean_squared_error(y_test, yhat))
```

```
The Mean Squared Error of the model is 8.31316268486917
```

In [164]:
```
print("The R Mean Squared Error of the model is", math.sqrt(mean_squared_error(y_test, yhat)))
```

```
The R Mean Squared Error of the model is 2.8832555705086516
```

In [165]:
```
# Intercept of the model
intercept = linear_regressor.intercept_

# Coefficeints of the model
m = linear_regressor.coef_
```

In [166]:

```
print("The intercept of the model is", intercept, "\n\nThe weights of the features are as follows:\n", m)
```

```
The intercept of the model is 199673.12053704684

The weights of the features are as follows:
 [-3.20715812e-02 -2.04798014e-02  9.15431201e-02  9.32558877e-02
  1.60775542e-05 -1.18534786e-02 -2.90322988e-05  4.31807766e-02
 -6.85738959e-02  2.23487580e-02  7.68092770e-02  4.30157311e-02
 -4.75479693e-01  4.86038167e-05  7.33769351e-10  9.54822995e-03
 -9.16181345e-02  5.93943805e+00  6.94182147e-01 -3.18380449e+05
 -2.01723307e+05  2.16992830e+03  2.16958476e+03  2.16887124e+03
  2.16977590e+03  2.16914807e+03  2.16888231e+03  2.16876489e+03
  1.18826343e+05  1.18825849e+05  2.16909364e+03  2.17020312e+03
  2.17001040e+03  2.16851799e+03  2.17065351e+03  2.17004570e+03
  1.18827833e+05  2.17059922e+03  2.16989482e+03  2.17047721e+03
  2.16972331e+03  2.17016721e+03  2.17124476e+03  2.17047939e+03
  2.16885901e+03  1.18826091e+05  2.16869723e+03  2.16956072e+03
  2.16797177e+03  2.16915735e+03  2.16880001e+03  2.16848294e+03
  2.16884116e+03  2.17019189e+03  2.16950918e+03  2.16947285e+03
  2.16642359e+03  2.16900722e+03  2.17063617e+03  2.17021969e+03
  1.18827401e+05  2.16971036e+03  1.18826214e+05  1.18825233e+05
  2.16898926e+03  2.16844631e+03  2.17115949e+03  1.18827444e+05
  2.16963726e+03  2.16818376e+03  2.16802926e+03  2.16681671e+03
  2.16727261e+03  2.16748332e+03  2.16868263e+03  2.16826762e+03
  2.16745452e+03  2.16800432e+03  2.16769945e+03  2.16844795e+03
  2.16910110e+03  2.16914631e+03  2.16855869e+03  1.18824990e+05
  2.16856201e+03  2.16835213e+03  2.17200027e+03  2.16991594e+03
  2.16809170e+03  2.16762418e+03  2.16817679e+03  2.16700281e+03
  2.16717697e+03  1.18824545e+05  1.18827150e+05  2.16945869e+03
  2.16846623e+03  2.17066667e+03  2.16930260e+03  1.18826164e+05
  2.17043461e+03  1.18825518e+05  2.17001207e+03  1.18828080e+05
  2.16919594e+03  2.16852756e+03  2.16922777e+03  2.16971917e+03
  2.16938384e+03  2.16925451e+03  2.16857990e+03  1.18826921e+05
  2.16999131e+03  2.16787513e+03  2.16899130e+03  2.17068234e+03
  1.18825497e+05  1.18826964e+05  2.16806527e+03  2.16867295e+03
  2.17008695e+03  2.16803738e+03  2.16708830e+03  1.18825162e+05
  2.16658882e+03  2.16627979e+03  2.16846368e+03  2.16781526e+03
  2.16702554e+03  2.16946289e+03  2.16954914e+03  2.17109144e+03
  2.16961113e+03  2.17009186e+03  2.16990630e+03  1.18827589e+05
  1.18827266e+05  2.16822930e+03  2.16935024e+03  2.16912302e+03
  1.18825851e+05  2.16956416e+03  2.17022601e+03  2.17009413e+03
  2.17114685e+03  2.16918728e+03  2.16906738e+03  2.16879851e+03
  1.18825129e+05  1.18826684e+05  2.16824461e+03  2.16787666e+03
  2.16986038e+03  1.18827263e+05  2.16837901e+03  2.17044041e+03
  2.16963167e+03  2.16906490e+03  2.17075757e+03  2.17110481e+03
  2.17119573e+03  2.16979956e+03  2.16948012e+03  2.17035024e+03
  2.16960022e+03  1.18826870e+05  1.18825401e+05  1.18826866e+05
  2.17127274e+03  2.16996059e+03  2.16998505e+03  2.17028118e+03
  1.18825159e+05  2.16895375e+03  2.16967438e+03  2.17092925e+03
  2.16980103e+03  1.18826490e+05  1.18826944e+05  2.16882894e+03
  2.17095943e+03  2.16962722e+03  2.16802887e+03  2.16841266e+03
  2.16798328e+03  2.16789270e+03  2.16699305e+03  2.16863527e+03
  2.17017040e+03  2.16974843e+03  2.16876126e+03  2.16804349e+03
  2.16822345e+03  1.18824208e+05  2.16658182e+03  1.18824976e+05
  2.16703640e+03  2.16816554e+03  2.16696932e+03  2.16902185e+03
  2.17029609e+03  2.16752145e+03  2.17080398e+03  2.17020757e+03]
```

Equation of the model

In [167]:
```python
# Equation of the model

def predictLE(weights, x):

  equation = 0
  for i in range(len(m)):
      equation += (weights[i]*x[i])
  equation += intercept
  # print("The Life Expectancy is " + str(equation))
  return equation
```

Training a Ridge Regression model

In [168]:
```python
# Fitting the Data
ridge = Ridge()
ridge.fit(X_train, y_train)

# Predicting the Target Variable
y_pred2 = ridge.predict(X_test)
```

```
C:\Users\joanj\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:212: LinAlgWarning: Ill-conditioned matrix (rcond=1.48442e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

Regression Plot

In [169]:

```python
# Plotting the true and predicted value
fig = plt.figure(figsize =(20, 10))
sns.regplot(y_test, y_pred2)
plt.xlabel("LE Test")
plt.ylabel("LE Predicted")
plt.title("True Value vs Predicted Value")
plt.show()
```

```
C:\Users\joanj\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Accuracy Count

In [170]:
```
# Chcking the accuracy of the model
print("The accuracy of the model is", ridge.score(X_test, y_test))
```

The accuracy of the model is 0.8139971860712808

In [171]:
```
print("The r2 score of the model is", r2_score(y_test, y_pred2))
```

The r2 score of the model is 0.8139971860712808

Evaluation Metrics

In [172]:
```
# Evaluation Metrics
print("The Absolute Mean Error of the model is", mean_absolute_error(y_test, y_pred2))
```

The Absolute Mean Error of the model is 3.0948610206583105

In [173]:
```
print("The Mean Squared Error of the model is", mean_squared_error(y_test, y_pred2))
```

The Mean Squared Error of the model is 16.695186985172498

In [174]:
```
print("The RMSE of the model is", math.sqrt(mean_squared_error(y_test, y_pred2)))
```

The RMSE of the model is 4.0859744229709145

In [175]:
```
# Intercept of the model
intercept2 = linear_regressor.intercept_

# Coefficeints of the model
m2 = linear_regressor.coef_
```

In [176]:

```python
print("The intercept of the model is", intercept2, "\n\nThe weights of the features are as follows:\n", m2)
```

```
The intercept of the model is 199673.12053704684

The weights of the features are as follows:
 [-3.20715812e-02 -2.04798014e-02  9.15431201e-02  9.32558877e-02
  1.60775542e-05 -1.18534786e-02 -2.90322988e-05  4.31807766e-02
 -6.85738959e-02  2.23487580e-02  7.68092770e-02  4.30157311e-02
 -4.75479693e-01  4.86038167e-05  7.33769351e-10  9.54822995e-03
 -9.16181345e-02  5.93943805e+00  6.94182147e-01 -3.18380449e+05
 -2.01723307e+05  2.16992830e+03  2.16958476e+03  2.16887124e+03
  2.16977590e+03  2.16914807e+03  2.16888231e+03  2.16876489e+03
  1.18826343e+05  1.18825849e+05  2.16909364e+03  2.17020312e+03
  2.17001040e+03  2.16851799e+03  2.17065351e+03  2.17004570e+03
  1.18827833e+05  2.17059922e+03  2.16989482e+03  2.17047721e+03
  2.16972331e+03  2.17016721e+03  2.17124476e+03  2.17047939e+03
  2.16885901e+03  1.18826091e+05  2.16869723e+03  2.16956072e+03
  2.16797177e+03  2.16915735e+03  2.16880001e+03  2.16848294e+03
  2.16884116e+03  2.17019189e+03  2.16950918e+03  2.16947285e+03
  2.16642359e+03  2.16900722e+03  2.17063617e+03  2.17021969e+03
  1.18827401e+05  2.16971036e+03  1.18826214e+05  1.18825233e+05
  2.16898926e+03  2.16844631e+03  2.17115949e+03  1.18827444e+05
  2.16963726e+03  2.16818376e+03  2.16802926e+03  2.16681671e+03
  2.16727261e+03  2.16748332e+03  2.16868263e+03  2.16826762e+03
  2.16745452e+03  2.16800432e+03  2.16769945e+03  2.16844795e+03
  2.16910110e+03  2.16914631e+03  2.16855869e+03  1.18824990e+05
  2.16856201e+03  2.16835213e+03  2.17200027e+03  2.16991594e+03
  2.16809170e+03  2.16762418e+03  2.16817679e+03  2.16700281e+03
  2.16717697e+03  1.18824545e+05  1.18827150e+05  2.16945869e+03
  2.16846623e+03  2.17066667e+03  2.16930260e+03  1.18826164e+05
  2.17043461e+03  1.18825518e+05  2.17001207e+03  1.18828080e+05
  2.16919594e+03  2.16852756e+03  2.16922777e+03  2.16971917e+03
  2.16938384e+03  2.16925451e+03  2.16857990e+03  1.18826921e+05
  2.16999131e+03  2.16787513e+03  2.16899130e+03  2.17068234e+03
  1.18825497e+05  1.18826964e+05  2.16806527e+03  2.16867295e+03
  2.17008695e+03  2.16803738e+03  2.16708830e+03  1.18825162e+05
  2.16658882e+03  2.16627979e+03  2.16846368e+03  2.16781526e+03
  2.16702554e+03  2.16946289e+03  2.16954914e+03  2.17109144e+03
  2.16961113e+03  2.17009186e+03  2.16990630e+03  1.18827589e+05
  1.18827266e+05  2.16822930e+03  2.16935024e+03  2.16912302e+03
  1.18825851e+05  2.16956416e+03  2.17022601e+03  2.17009413e+03
  2.17114685e+03  2.16918728e+03  2.16906738e+03  2.16879851e+03
  1.18825129e+05  1.18826684e+05  2.16824461e+03  2.16787666e+03
  2.16986038e+03  1.18827263e+05  2.16837901e+03  2.17044041e+03
  2.16963167e+03  2.16906490e+03  2.17075757e+03  2.17110481e+03
  2.17119573e+03  2.16979956e+03  2.16948012e+03  2.17035024e+03
  2.16960022e+03  1.18826870e+05  1.18825401e+05  1.18826866e+05
  2.17127274e+03  2.16996059e+03  2.16998505e+03  2.17028118e+03
  1.18825159e+05  2.16895375e+03  2.16967438e+03  2.17092925e+03
  2.16980103e+03  1.18826490e+05  1.18826944e+05  2.16882894e+03
  2.17095943e+03  2.16962722e+03  2.16802887e+03  2.16841266e+03
  2.16798328e+03  2.16789270e+03  2.16699305e+03  2.16863527e+03
  2.17017040e+03  2.16974843e+03  2.16876126e+03  2.16804349e+03
  2.16822345e+03  1.18824208e+05  2.16658182e+03  1.18824976e+05
  2.16703640e+03  2.16816554e+03  2.16696932e+03  2.16902185e+03
  2.17029609e+03  2.16752145e+03  2.17080398e+03  2.17020757e+03]
```

In [177]:

```
# Equation of the model

def predictLE2(weights, x):

  equation = 0
  for i in range(len(m2)):
      equation += (weights[i]*x[i])
  equation += intercept2
  print("The Life Expectancy is " + str(equation))
```

Training a Decision Tree model

In [178]:

```
# Fitting the Data
tree = DecisionTreeRegressor(max_depth=4)
tree.fit(X_train, y_train)

# Predicting the Target Variable
y_pred3 = tree.predict(X_test)
```

Regression Plot

In [179]:

```
# Plotting the true and predicted value
fig = plt.figure(figsize =(20, 10))
sns.regplot(y_test, y_pred3)
plt.xlabel("LE Test")
plt.ylabel("LE Predicted")
plt.title("True Value vs Predicted Value")
plt.show()
```

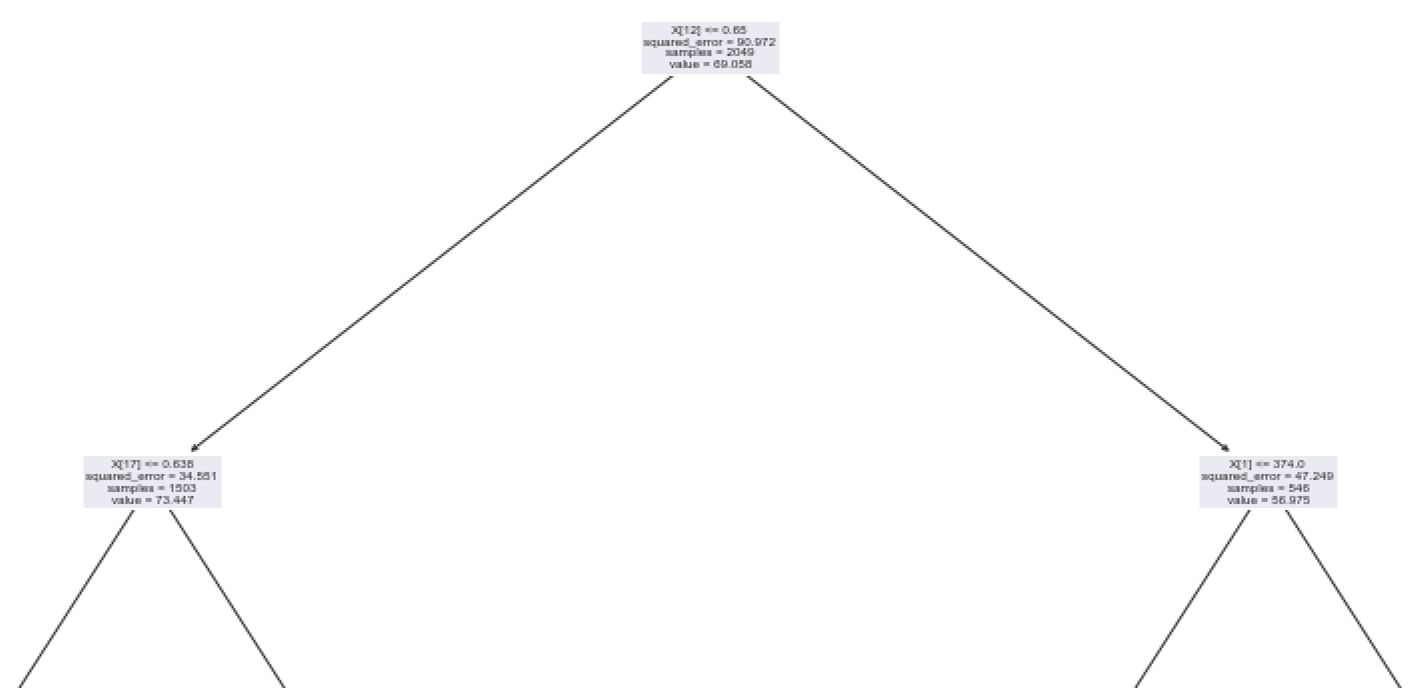
```
C:\Users\joanj\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

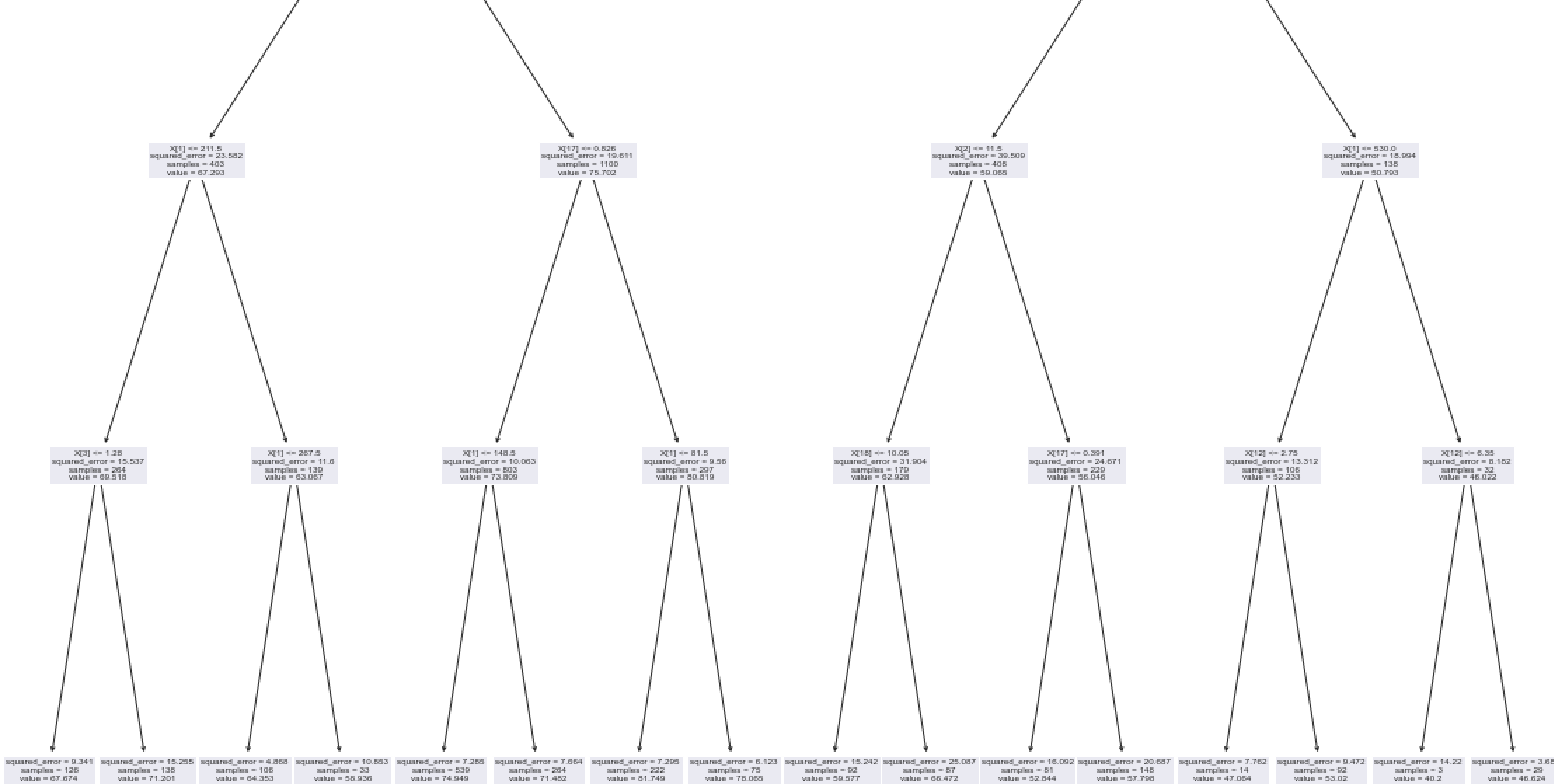
In [180]:

```python
# Show the decision tree
plt.figure(figsize=(20,20))
plot_tree(tree)
```

Out[180]:

```
[Text(0.5, 0.9, 'X[12] <= 0.65\nsquared_error = 90.972\nsamples = 2049\nvalue = 69.058'),
 Text(0.25, 0.7, 'X[17] <= 0.638\nsquared_error = 34.551\nsamples = 1503\nvalue = 73.447'),
 Text(0.125, 0.5, 'X[1] <= 211.5\nsquared_error = 23.582\nsamples = 403\nvalue = 67.293'),
 Text(0.0625, 0.3, 'X[3] <= 1.28\nsquared_error = 15.537\nsamples = 264\nvalue = 69.518'),
 Text(0.03125, 0.1, 'squared_error = 9.341\nsamples = 126\nvalue = 67.674'),
 Text(0.09375, 0.1, 'squared_error = 15.255\nsamples = 138\nvalue = 71.201'),
 Text(0.1875, 0.3, 'X[1] <= 267.5\nsquared_error = 11.6\nsamples = 139\nvalue = 63.067'),
 Text(0.15625, 0.1, 'squared_error = 4.868\nsamples = 106\nvalue = 64.353'),
 Text(0.21875, 0.1, 'squared_error = 10.853\nsamples = 33\nvalue = 58.936'),
 Text(0.375, 0.5, 'X[17] <= 0.826\nsquared_error = 19.611\nsamples = 1100\nvalue = 75.702'),
 Text(0.3125, 0.3, 'X[1] <= 148.5\nsquared_error = 10.063\nsamples = 803\nvalue = 73.809'),
 Text(0.28125, 0.1, 'squared_error = 7.285\nsamples = 539\nvalue = 74.949'),
 Text(0.34375, 0.1, 'squared_error = 7.664\nsamples = 264\nvalue = 71.482'),
 Text(0.4375, 0.3, 'X[1] <= 81.5\nsquared_error = 9.56\nsamples = 297\nvalue = 80.819'),
 Text(0.40625, 0.1, 'squared_error = 7.295\nsamples = 222\nvalue = 81.749'),
 Text(0.46875, 0.1, 'squared_error = 6.123\nsamples = 75\nvalue = 78.065'),
 Text(0.75, 0.7, 'X[1] <= 374.0\nsquared_error = 47.249\nsamples = 546\nvalue = 56.975'),
 Text(0.625, 0.5, 'X[2] <= 11.5\nsquared_error = 39.509\nsamples = 408\nvalue = 59.065'),
 Text(0.5625, 0.3, 'X[18] <= 10.05\nsquared_error = 31.904\nsamples = 179\nvalue = 62.928'),
 Text(0.53125, 0.1, 'squared_error = 15.242\nsamples = 92\nvalue = 59.577'),
 Text(0.59375, 0.1, 'squared_error = 25.087\nsamples = 87\nvalue = 66.472'),
 Text(0.6875, 0.3, 'X[17] <= 0.391\nsquared_error = 24.671\nsamples = 229\nvalue = 56.046'),
 Text(0.65625, 0.1, 'squared_error = 16.092\nsamples = 81\nvalue = 52.844'),
 Text(0.71875, 0.1, 'squared_error = 20.687\nsamples = 148\nvalue = 57.798'),
 Text(0.875, 0.5, 'X[1] <= 530.0\nsquared_error = 18.994\nsamples = 138\nvalue = 50.793'),
 Text(0.8125, 0.3, 'X[12] <= 2.75\nsquared_error = 13.312\nsamples = 106\nvalue = 52.233'),
 Text(0.78125, 0.1, 'squared_error = 7.762\nsamples = 14\nvalue = 47.064'),
 Text(0.84375, 0.1, 'squared_error = 9.472\nsamples = 92\nvalue = 53.02'),
 Text(0.9375, 0.3, 'X[12] <= 6.35\nsquared_error = 8.182\nsamples = 32\nvalue = 46.022'),
 Text(0.90625, 0.1, 'squared_error = 14.22\nsamples = 3\nvalue = 40.2'),
 Text(0.96875, 0.1, 'squared_error = 3.688\nsamples = 29\nvalue = 46.624')]
```





Accuracy Count

In [181]:
```
# Chcking the accuracy of the model
print("The accuracy of the model is", tree.score(X_test, y_test))
```

```
The accuracy of the model is 0.8569892944426825
```

In [182]:
```
print("The r2 score of the model is", r2_score(y_test, y_pred3))
```

```
The r2 score of the model is 0.8569892944426825
```

Evaluation Metrics

In [183]:
```
# Evaluation Metrics
print("The Absolute Mean Error of the model is", mean_absolute_error(y_test, y_pred3))
```

```
The Absolute Mean Error of the model is 2.693555539173137
```

In [184]:
```
print("The Mean Squared Error of the model is", mean_squared_error(y_test, y_pred3))
```

```
The Mean Squared Error of the model is 12.836313708005761
```

In [185]:
```
print("The RMSE of the model is", math.sqrt(mean_squared_error(y_test, y_pred3)))
```

```
The RMSE of the model is 3.58278016462157
```

In [186]:
```
# Intercept of the model
intercept3 = linear_regressor.intercept_

# Coefficeints of the model
m3 = linear_regressor.coef_
```

In [187]:
```
print("The intercept of the model is", intercept3, "\n\nThe weights of the features are as follows:\n", m3)
```

```
The intercept of the model is 199673.12053704684

The weights of the features are as follows:
 [-3.20715812e-02 -2.04798014e-02  9.15431201e-02  9.32558877e-02
  1.60775542e-05 -1.18534786e-02 -2.90322988e-05  4.31807766e-02
 -6.85738959e-02  2.23487580e-02  7.68092770e-02  4.30157311e-02
 -4.75479693e-01  4.86038167e-05  7.33769351e-10  9.54822995e-03
 -9.16181345e-02  5.93943805e+00  6.94182147e-01 -3.18380449e+05
 -2.01723307e+05  2.16992830e+03  2.16958476e+03  2.16887124e+03
  2.16977590e+03  2.16914807e+03  2.16888231e+03  2.16876489e+03
  1.18826343e+05  1.18825849e+05  2.16909364e+03  2.17020312e+03
  2.17001040e+03  2.16851799e+03  2.17065351e+03  2.17004570e+03
  1.18827833e+05  2.17059922e+03  2.16989482e+03  2.17047721e+03
  2.16972331e+03  2.17016721e+03  2.17124476e+03  2.17047939e+03
  2.16885901e+03  1.18826091e+05  2.16869723e+03  2.16956072e+03
  2.16797177e+03  2.16915735e+03  2.16880001e+03  2.16848294e+03
  2.16884116e+03  2.17019189e+03  2.16950918e+03  2.16947285e+03
  2.16642359e+03  2.16900722e+03  2.17063617e+03  2.17021969e+03
  1.18827401e+05  2.16971036e+03  1.18826214e+05  1.18825233e+05
  2.16898926e+03  2.16844631e+03  2.17115949e+03  1.18827444e+05
  2.16963726e+03  2.16818376e+03  2.16802926e+03  2.16681671e+03
  2.16727261e+03  2.16748332e+03  2.16868263e+03  2.16826762e+03
  2.16745452e+03  2.16800432e+03  2.16769945e+03  2.16844795e+03
  2.16910110e+03  2.16914631e+03  2.16855869e+03  1.18824990e+05
  2.16856201e+03  2.16835213e+03  2.17200027e+03  2.16991594e+03
  2.16809170e+03  2.16762418e+03  2.16817679e+03  2.16700281e+03
  2.16717697e+03  1.18824545e+05  1.18827150e+05  2.16945869e+03
  2.16846623e+03  2.17066667e+03  2.16930260e+03  1.18826164e+05
  2.17043461e+03  1.18825518e+05  2.17001207e+03  1.18828080e+05
  2.16919594e+03  2.16852756e+03  2.16922777e+03  2.16971917e+03
  2.16938384e+03  2.16925451e+03  2.16857990e+03  1.18826921e+05
  2.16999131e+03  2.16787513e+03  2.16899130e+03  2.17068234e+03
  1.18825497e+05  1.18826964e+05  2.16806527e+03  2.16867295e+03
  2.17008695e+03  2.16803738e+03  2.16708830e+03  1.18825162e+05
  2.16658882e+03  2.16627979e+03  2.16846368e+03  2.16781526e+03
  2.16702554e+03  2.16946289e+03  2.16954914e+03  2.17109144e+03
  2.16961113e+03  2.17009186e+03  2.16990630e+03  1.18827589e+05
  1.18827266e+05  2.16822930e+03  2.16935024e+03  2.16912302e+03
  1.18825851e+05  2.16956416e+03  2.17022601e+03  2.17009413e+03
  2.17114685e+03  2.16918728e+03  2.16906738e+03  2.16879851e+03
  1.18825129e+05  1.18826684e+05  2.16824461e+03  2.16787666e+03
  2.16986038e+03  1.18827263e+05  2.16837901e+03  2.17044041e+03
  2.16963167e+03  2.16906490e+03  2.17075757e+03  2.17110481e+03
  2.17119573e+03  2.16979956e+03  2.16948012e+03  2.17035024e+03
  2.16960022e+03  1.18826870e+05  1.18825401e+05  1.18826866e+05
  2.17127274e+03  2.16996059e+03  2.16998505e+03  2.17028118e+03
  1.18825159e+05  2.16895375e+03  2.16967438e+03  2.17092925e+03
  2.16980103e+03  1.18826490e+05  1.18826944e+05  2.16882894e+03
  2.17095943e+03  2.16962722e+03  2.16802887e+03  2.16841266e+03
  2.16798328e+03  2.16789270e+03  2.16699305e+03  2.16863527e+03
  2.17017040e+03  2.16974843e+03  2.16876126e+03  2.16804349e+03
  2.16822345e+03  1.18824208e+05  2.16658182e+03  1.18824976e+05
  2.16703640e+03  2.16816554e+03  2.16696932e+03  2.16902185e+03
  2.17029609e+03  2.16752145e+03  2.17080398e+03  2.17020757e+03]
```

In [188]:

```
# Equation of the model

def predictLE3(weights, x):

  equation = 0
  for i in range(len(m3)):
      equation += (weights[i]*x[i])
  equation += intercept3
  print("The Life Expectancy is " + str(equation))
```

Training a Random Forest model

In [189]:

```
# Fitting the Data
forest = RandomForestRegressor(n_estimators=100)
forest.fit(X_train, y_train)

# Predicting the Target Variable
y_pred4 = forest.predict(X_test)
```

Regression Plot

In [190]:

```
# Plotting the true and predicted value
fig = plt.figure(figsize =(20, 10))
sns.regplot(y_test, y_pred4)
plt.xlabel("LE Test")
plt.ylabel("LE Predicted")
plt.title("True Value vs Predicted Value")
plt.show()
```

```
C:\Users\joanj\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Accuracy Count

In [191]:
```
# Chcking the accuracy of the model
print("The accuracy of the model is", forest.score(X_test, y_test))
```

```
The accuracy of the model is 0.9558929929973955
```

In [192]:
```
print("The r2 score of the model is", r2_score(y_test, y_pred4))
```

```
The r2 score of the model is 0.9558929929973955
```

Evaluation Metrics

In [193]:
```
# Evaluation Metrics
print("The Absolute Mean Error of the model is", mean_absolute_error(y_test, y_pred4))
```

```
The Absolute Mean Error of the model is 1.2947281001137665
```

In [194]:
```
print("The Mean Squared Error of the model is", mean_squared_error(y_test, y_pred4))
```

```
The Mean Squared Error of the model is 3.9589440273037573
```

In [195]:
```
print("The Mean Squared Error of the model is", math.sqrt(mean_squared_error(y_test, y_pred4)))
```

```
The Mean Squared Error of the model is 1.989709533400229
```

In [196]:
```
# Intercept of the model
intercept4 = linear_regressor.intercept_

# Coefficeints of the model
m4 = linear_regressor.coef_
```

In [197]:
```
print("The intercept of the model is", intercept4, "\n\nThe weights of the features are as follows:\n", m4)
```

```
The intercept of the model is 199673.12053704684

The weights of the features are as follows:
 [-3.20715812e-02 -2.04798014e-02  9.15431201e-02  9.32558877e-02
  1.60775542e-05 -1.18534786e-02 -2.90322988e-05  4.31807766e-02
 -6.85738959e-02  2.23487580e-02  7.68092770e-02  4.30157311e-02
 -4.75479693e-01  4.86038167e-05  7.33769351e-10  9.54822995e-03
 -9.16181345e-02  5.93943805e+00  6.94182147e-01 -3.18380449e+05
 -2.01723307e+05  2.16992830e+03  2.16958476e+03  2.16887124e+03
  2.16977590e+03  2.16914807e+03  2.16888231e+03  2.16876489e+03
  1.18826343e+05  1.18825849e+05  2.16909364e+03  2.17020312e+03
  2.17001040e+03  2.16851799e+03  2.17065351e+03  2.17004570e+03
  1.18827833e+05  2.17059922e+03  2.16989482e+03  2.17047721e+03
  2.16972331e+03  2.17016721e+03  2.17124476e+03  2.17047939e+03
  2.16885901e+03  1.18826091e+05  2.16869723e+03  2.16956072e+03
  2.16797177e+03  2.16915735e+03  2.16880001e+03  2.16848294e+03
  2.16884116e+03  2.17019189e+03  2.16950918e+03  2.16947285e+03
  2.16642359e+03  2.16900722e+03  2.17063617e+03  2.17021969e+03
  1.18827401e+05  2.16971036e+03  1.18826214e+05  1.18825233e+05
  2.16898926e+03  2.16844631e+03  2.17115949e+03  1.18827444e+05
  2.16963726e+03  2.16818376e+03  2.16802926e+03  2.16681671e+03
  2.16727261e+03  2.16748332e+03  2.16868263e+03  2.16826762e+03
  2.16745452e+03  2.16800432e+03  2.16769945e+03  2.16844795e+03
  2.16910110e+03  2.16914631e+03  2.16855869e+03  1.18824990e+05
  2.16856201e+03  2.16835213e+03  2.17200027e+03  2.16991594e+03
  2.16809170e+03  2.16762418e+03  2.16817679e+03  2.16700281e+03
  2.16717697e+03  1.18824545e+05  1.18827150e+05  2.16945869e+03
  2.16846623e+03  2.17066667e+03  2.16930260e+03  1.18826164e+05
  2.17043461e+03  1.18825518e+05  2.17001207e+03  1.18828080e+05
  2.16919594e+03  2.16852756e+03  2.16922777e+03  2.16971917e+03
  2.16938384e+03  2.16925451e+03  2.16857990e+03  1.18826921e+05
  2.16999131e+03  2.16787513e+03  2.16899130e+03  2.17068234e+03
  1.18825497e+05  1.18826964e+05  2.16806527e+03  2.16867295e+03
  2.17008695e+03  2.16803738e+03  2.16708830e+03  1.18825162e+05
  2.16658882e+03  2.16627979e+03  2.16846368e+03  2.16781526e+03
  2.16702554e+03  2.16946289e+03  2.16954914e+03  2.17109144e+03
  2.16961113e+03  2.17009186e+03  2.16990630e+03  1.18827589e+05
  1.18827266e+05  2.16822930e+03  2.16935024e+03  2.16912302e+03
  1.18825851e+05  2.16956416e+03  2.17022601e+03  2.17009413e+03
  2.17114685e+03  2.16918728e+03  2.16906738e+03  2.16879851e+03
  1.18825129e+05  1.18826684e+05  2.16824461e+03  2.16787666e+03
  2.16986038e+03  1.18827263e+05  2.16837901e+03  2.17044041e+03
  2.16963167e+03  2.16906490e+03  2.17075757e+03  2.17110481e+03
  2.17119573e+03  2.16979956e+03  2.16948012e+03  2.17035024e+03
  2.16960022e+03  1.18826870e+05  1.18825401e+05  1.18826866e+05
  2.17127274e+03  2.16996059e+03  2.16998505e+03  2.17028118e+03
  1.18825159e+05  2.16895375e+03  2.16967438e+03  2.17092925e+03
  2.16980103e+03  1.18826490e+05  1.18826944e+05  2.16882894e+03
  2.17095943e+03  2.16962722e+03  2.16802887e+03  2.16841266e+03
  2.16798328e+03  2.16789270e+03  2.16699305e+03  2.16863527e+03
  2.17017040e+03  2.16974843e+03  2.16876126e+03  2.16804349e+03
  2.16822345e+03  1.18824208e+05  2.16658182e+03  1.18824976e+05
  2.16703640e+03  2.16816554e+03  2.16696932e+03  2.16902185e+03
  2.17029609e+03  2.16752145e+03  2.17080398e+03  2.17020757e+03]
```

In [198]:

```
# Equation of the model

def predictLE4(weights, x):

  equation = 0
  for i in range(len(m4)):
      equation += (weights[i]*x[i])
  equation += intercept4
  print("The Life Expectancy is " + str(equation))
```

Training a SGD Regression model:

In [199]:

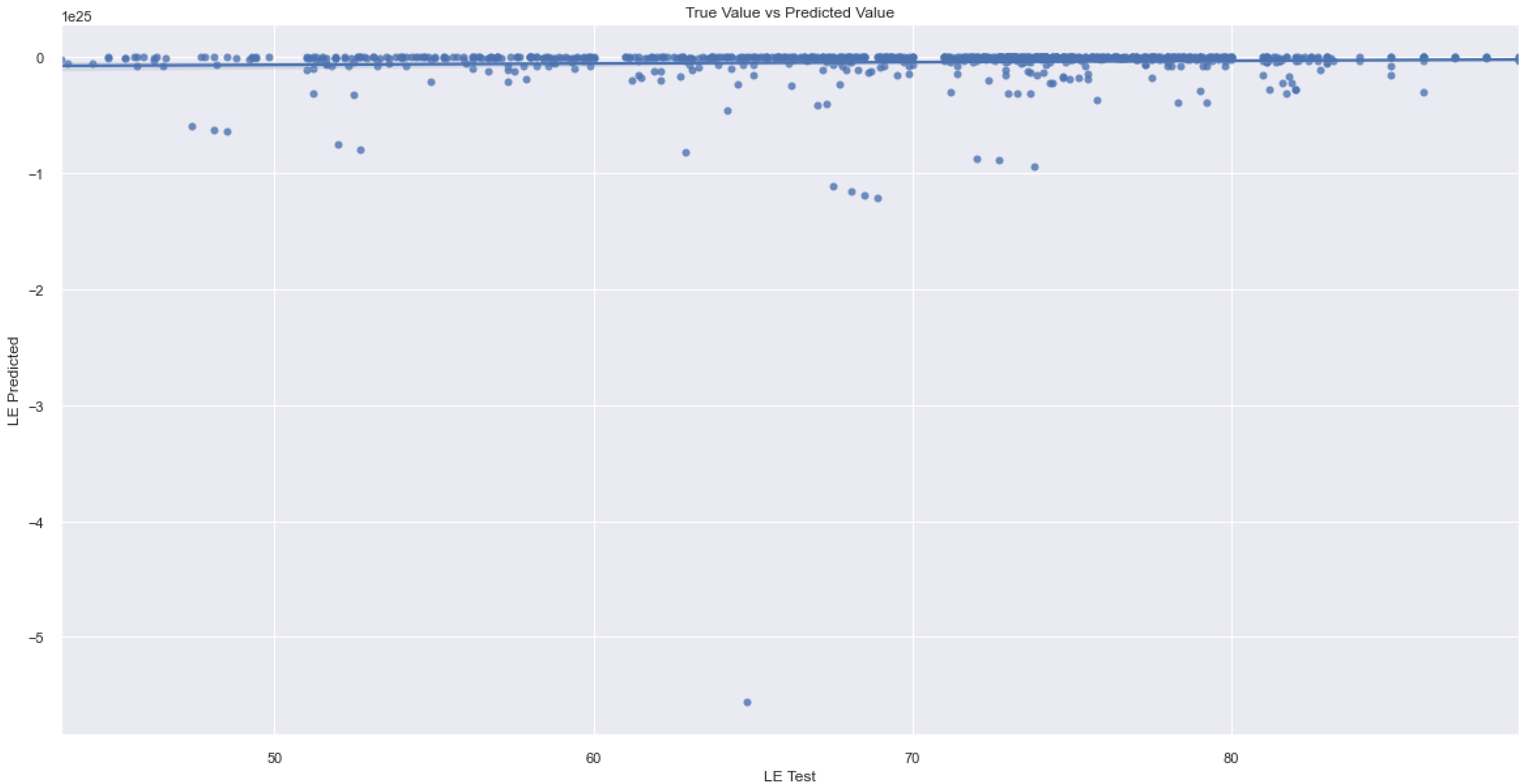
```
# Fitting the Data
sgdr = SGDRegressor()
sgdr.fit(X_train, y_train)

# Predicting the Target Variable
y_pred5 = sgdr.predict(X_test)
```

Regression Plot

In [200]:

```python
# Plotting the true and predicted value
fig = plt.figure(figsize =(20, 10))
sns.regplot(y_test, y_pred5)
plt.xlabel("LE Test")
plt.ylabel("LE Predicted")
plt.title("True Value vs Predicted Value")
plt.show()
```

```
C:\Users\joanj\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [201]:
```python
# Chcking the accuracy of the model
print("The accuracy of the model is", sgdr.score(X_train, y_train))
```

```
The accuracy of the model is -8.436743678238087e+46
```

In [202]:
```python
cv_score = cross_val_score(sgdr, X, y, cv = 10)
print("CV mean score: ", cv_score.mean())
```

```
CV mean score:  -1.0706117933028202e+48
```

Boosting Efficiency

In [203]:
```python
#generic function to fit model and return metrics for every algorithm
def boost_models(x):
    #transforming target variable through quantile transformer
    regr_trans = TransformedTargetRegressor(regressor=x, transformer=QuantileTransformer(output_distribution='normal'))
    regr_trans.fit(X_train, y_train)
    yhat = regr_trans.predict(X_test)
    algoname= x.__class__.__name__
    return algoname, round(r2_score(y_test, yhat),3), round(mean_absolute_error(y_test, yhat),2), round(np.sqrt(mean_squared_error(y_test, yhat)),2)

algo=[GradientBoostingRegressor(), lightgbm.LGBMRegressor(), xgboost.XGBRFRegressor()]
score=[]

for a in algo:
    score.append(boost_models(a))

#Collate all scores in a table
pd.DataFrame(score, columns=['Model','Score','MAE','RMSE'])
```

Out[203]:

	Model	Score	MAE	RMSE
0	GradientBoostingRegressor	0.930	1.83	2.50
1	LGBMRegressor	0.955	1.37	2.00
2	XGBRFRegressor	0.924	1.86	2.61

FLASK Attempt:

In [74]:
```
import pickle
from joblib import dump, load
dump(linear_regressor, 'model.joblib')
```

Out[74]: ['model.joblib']

In [75]:
```
loaded_model = load('model.joblib')
```

In [76]:
```
# !pip3 install kaleido
```

In [77]:
```
def make_picture2(training_data, model, new_input_arr, output_file='predictions_pic.svg'):

  # x_new = [1,2,3,4,5,6,7,8,9,10]
  x_new=np.arange(10).reshape((10, 1))
  preds = model.predict(np.array(x_new).reshape(1,-1))
  fig = px.scatter(x=y_test, y=y_pred, title="Predicted Life Expectancy", labels={'x': 'True Values',
                                                                              'y': 'Predicted Value Life Expectancy'})
  x_new=np.array(x_new)

  fig.add_trace(
   go.Scatter(x=x_new.reshape(x_new.shape[0]), y=preds, mode='lines', name='Model'))

  if new_input_arr is not False:
    new_preds = model.predict(np.array(new_input_arr).reshape(1,-1))
    new_input_arr = np.array(new_input_arr)
    fig.add_trace(
      go.Scatter(x=new_input_arr.reshape(new_input_arr.shape[0]), y=new_preds, name='New Outputs', mode='markers', marker=dict(
            color='purple',
            size=10,
            line=dict(
                color='purple',
                width=2
            ))))
  # fig.write_image(output_file, width=800)
  return fig
```

In [78]:
```
def floats_string_to_np_arr(floats_str):
  def is_float(s):
      try:
        float(s)
        return True
      except:
        return False
  floats = [float(x) for x in floats_str.split(",") if is_float(x)]
  return floats
  # floats.reshape(len(np.array(floats)),1)
```

In [79]:
```
import pickle
```

In [80]:
```
pickle.dump(linear_regressor, open('pickle_file', 'wb'))
```