METRO COLLEGE
OF TECHNOLOGY

Mini Project for Developing and Comparing Regression and Classification Machine Learning Models in Python Programming

---

Submitted by:

Mohammad Monjur-E-Elahi

Course: Machine Learning and Big Data Analytics [DSA10]

Program: Data Science and Application - Advanced Diploma [6060]

Metro College of Technology

Date: 29 May, 2021

Source of the Dataset:

About this file
The Global Health Observatory (GHO) data repository under World Health Organization (WHO) keeps track of the health status as well as many other related factors for all countries The datasets are made available to public for the purpose of health data analysis. The dataset related to life expectancy, health factors for 193 countries has been collected from the same WHO data repository website and its corresponding economic data was collected from United Nation website. Among all categories of health-related factors only those critical factors were chosen which are more representative. It has been observed that in the past 15 years , there has been a huge development in health sector resulting in improvement of human mortality rates especially in the developing nations in comparison to the past 30 years. Therefore, in this project we have considered data from year 2000-2015 for 193 countries for further analysis. The individual data files have been merged together into a single dataset. On initial visual inspection of the data showed some missing values. As the datasets were from WHO, we found no evident errors. Missing data was handled in R software by using Missmap command. The result indicated that most of the missing data was for population, Hepatitis B and GDP. The missing data were from less known countries like Vanuatu, Tonga, Togo,Cabo Verde etc. Finding all data for these countries was difficult and hence, it was decided that we exclude these countries from the final model dataset. The final merged file(final dataset) consists of 22 Columns and 2938 rows which meant 20 predicting variables. All predicting variables was then divided into several broad categories:•Immunization related factors, Mortality factors, Economical factors and Social factors.
Source: https://www.kaggle.com

Regression Models

Let us import the required initial libraries.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
%matplotlib inline
```

Let us check the Current Working Directory.

In [2]:
```
os.getcwd()
```

Out[2]:
```
'C:\\Users\\ruzdomain\\Desktop\\MLBDA\\PROJECT'
```

Let us ensure that the Current Working Directory remains the one where we are working.

In [3]:
```
os.chdir(r'C:\Users\ruzdomain\Desktop\MLBDA\PROJECT')
```

In [4]:
```
os.getcwd()
```

Out[4]:
```
'C:\\Users\\ruzdomain\\Desktop\\MLBDA\\PROJECT'
```

Let us import the dataset by reading using panadas and explore the dataset.

In [5]:
```
df_reg = pd.read_csv('Life Expectancy Data.csv')
df_reg.head()
```

Out[5]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2

5 rows × 22 columns

In [6]:
```
df_reg.describe(include='all')
```

Out[6]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS
count	2938	2938.000000	2938	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	...	2919.000000	2712.00000	2919.000000	2938.000000
unique	193	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
top	Hungary	NaN	Developing	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
freq	16	NaN	2426	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
mean	NaN	2007.518720	NaN	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	...	82.550188	5.93819	82.324084	1.742103
std	NaN	4.613841	NaN	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	...	23.428046	2.49832	23.716912	5.077785
min	NaN	2000.000000	NaN	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	...	3.000000	0.37000	2.000000	0.100000
25%	NaN	2004.000000	NaN	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	...	78.000000	4.26000	78.000000	0.100000
50%	NaN	2008.000000	NaN	72.100000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	...	93.000000	5.75500	93.000000	0.100000
75%	NaN	2012.000000	NaN	75.700000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	...	97.000000	7.49250	97.000000	0.800000
max	NaN	2015.000000	NaN	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	...	99.000000	17.60000	99.000000	50.600000

11 rows × 22 columns

In [7]:
```
df_reg.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
```

In [8]:
```
df_reg.shape
```

Out[8]:
```
(2938, 22)
```

In [9]:
```
df_reg.dtypes
```

Out[9]:
```
Country                           object
Year                               int64
Status                            object
Life expectancy                  float64
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
dtype: object
```

Let us check the missing values per column and also in percentage.

In [10]:
```
df_reg.apply(lambda x: sum(x.isnull()))
```

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

In [11]:
```
def percentage_of_miss(df):
  df1=df[df.columns[df.isnull().sum()>=1]]
  total_miss = df1.isnull().sum().sort_values(ascending=False)
  percent_miss = (df1.isnull().sum()/df1.isnull().count()).sort_values(ascending=False)
  missing_data = pd.concat([total_miss, percent_miss], axis=1, keys=['Number of Missing', 'Percentage'])
  return(missing_data)
```

In [12]:
```
percentage_of_miss(df_reg)
```

Out[12]:

	Number of Missing	Percentage
Population	652	0.221920
Hepatitis B	553	0.188223
GDP	448	0.152485
Total expenditure	226	0.076923
Alcohol	194	0.066031
Income composition of resources	167	0.056841
Schooling	163	0.055480
thinness 5-9 years	34	0.011572
thinness 1-19 years	34	0.011572
BMI	34	0.011572
Diphtheria	19	0.006467
Polio	19	0.006467
Adult Mortality	10	0.003404
Life expectancy	10	0.003404

In [13]:
```
df_reg['Country'].value_counts()
```

Out[13]:
```
Hungary                16
Dominican Republic     16
Costa Rica             16
Congo                  16
Mongolia               16
                       ..
Saint Kitts and Nevis   1
Nauru                   1
Dominica                1
Niue                    1
San Marino              1
Name: Country, Length: 193, dtype: int64
```

In [14]:
```
df_reg['Status'].value_counts()
```

Out[14]:
```
Developing    2426
Developed      512
Name: Status, dtype: int64
```

Let us check duplicate rwos and missing values

In [16]:
```
dup_df_reg = df_reg[df_reg.duplicated()]
dup_df_reg
```

Out[16]:

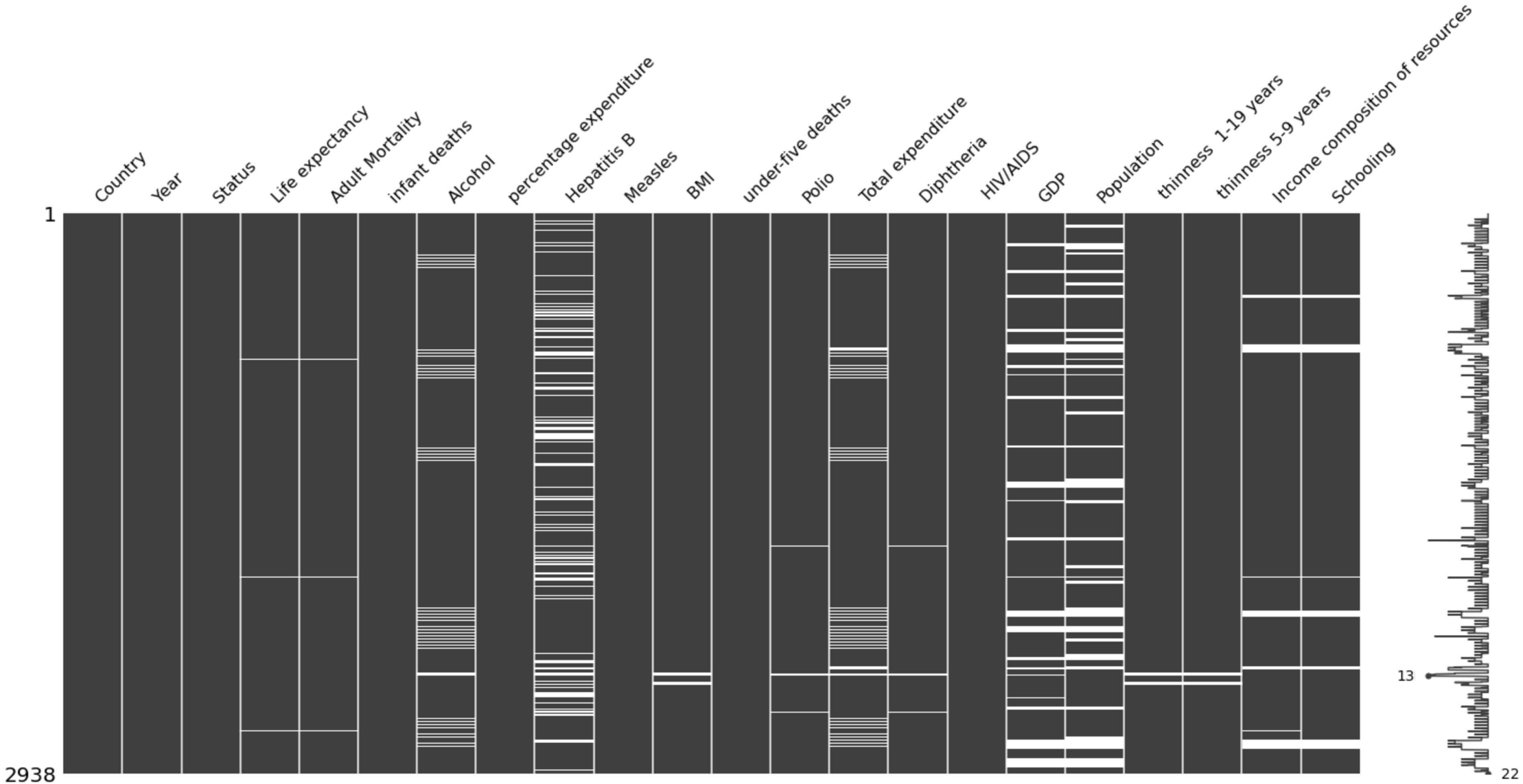
Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income composition of resources	S

0 rows × 22 columns

In [17]:
```
import missingno as msno
```

In [18]:
```
msno.matrix(df_reg)
```

Out[18]: <AxesSubplot:>



Have a look at the correlation among all the columns numerical.

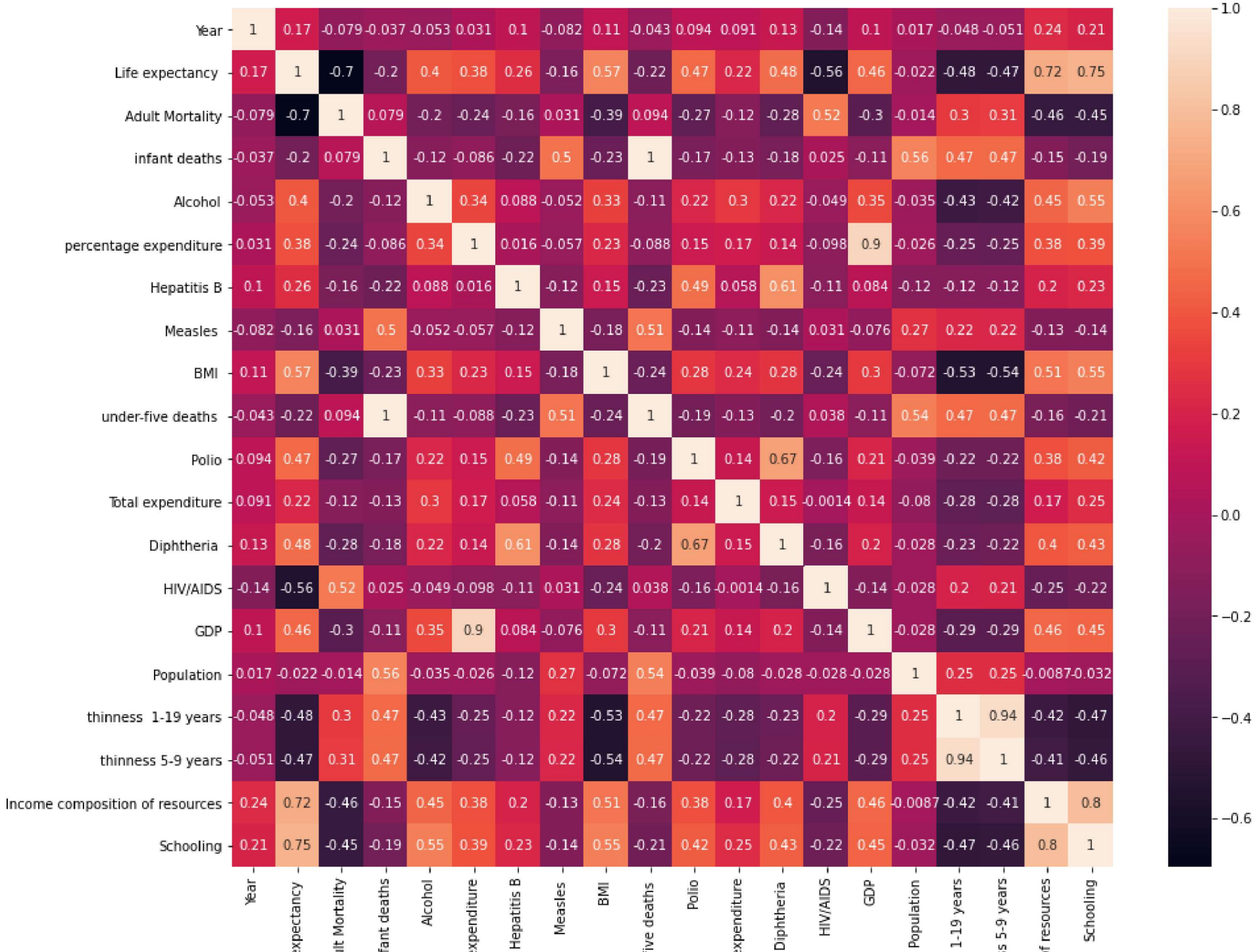
In [19]:
```
corr_df_reg=df_reg.corr()
corr_df_reg
```

Out[19]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	t
Year	1.000000	0.170033	-0.079052	-0.037415	-0.052990	0.031400	0.104333	-0.082493	0.108974	-0.042937	0.094158	0.090740	0.134337	-0.139741	0.101620	0.016969	-0
Life expectancy	0.170033	1.000000	-0.696359	-0.196557	0.404877	0.381864	0.256762	-0.157586	0.567694	-0.222529	0.465556	0.218086	0.479495	-0.556556	0.461455	-0.021538	-0
Adult Mortality	-0.079052	-0.696359	1.000000	0.078756	-0.195848	-0.242860	-0.162476	0.031176	-0.387017	0.094146	-0.274823	-0.115281	-0.275131	0.523821	-0.296049	-0.013647	0
infant deaths	-0.037415	-0.196557	0.078756	1.000000	-0.115638	-0.085612	-0.223566	0.501128	-0.227279	0.996629	-0.170689	-0.128616	-0.175171	0.025231	-0.108427	0.556801	0
Alcohol	-0.052990	0.404877	-0.195848	-0.115638	1.000000	0.341285	0.087549	-0.051827	0.330408	-0.112370	0.221734	0.296942	0.222020	-0.048845	0.354712	-0.035252	-0
percentage expenditure	0.031400	0.381864	-0.242860	-0.085612	0.341285	1.000000	0.016274	-0.056596	0.228700	-0.087852	0.147259	0.174420	0.143624	-0.097857	0.899373	-0.025662	-0
Hepatitis B	0.104333	0.256762	-0.162476	-0.223566	0.087549	0.016274	1.000000	-0.120529	0.150380	-0.233126	0.486171	0.058280	0.611495	-0.112675	0.083903	-0.123321	-0
Measles	-0.082493	-0.157586	0.031176	0.501128	-0.051827	-0.056596	-0.120529	1.000000	-0.175977	0.507809	-0.136166	-0.106241	-0.141882	0.030899	-0.076466	0.265966	0
BMI	0.108974	0.567694	-0.387017	-0.227279	0.330408	0.228700	0.150380	-0.175977	1.000000	-0.237669	0.284569	0.242503	0.283147	-0.243717	0.301557	-0.072301	-0
under-five deaths	-0.042937	-0.222529	0.094146	0.996629	-0.112370	-0.087852	-0.233126	0.507809	-0.237669	1.000000	-0.188720	-0.130148	-0.195668	0.038062	-0.112081	0.544423	0
Polio	0.094158	0.465556	-0.274823	-0.170689	0.221734	0.147259	0.486171	-0.136166	0.284569	-0.188720	1.000000	0.137330	0.673553	-0.159560	0.211976	-0.038540	-0
Total expenditure	0.090740	0.218086	-0.115281	-0.128616	0.296942	0.174420	0.058280	-0.106241	0.242503	-0.130148	0.137330	1.000000	0.152754	-0.001389	0.138364	-0.079662	-0
Diphtheria	0.134337	0.479495	-0.275131	-0.175171	0.222020	0.143624	0.611495	-0.141882	0.283147	-0.195668	0.673553	0.152754	1.000000	-0.164860	0.200666	-0.028444	-0
HIV/AIDS	-0.139741	-0.556556	0.523821	0.025231	-0.048845	-0.097857	-0.112675	0.030899	-0.243717	0.038062	-0.159560	-0.001389	-0.164860	1.000000	-0.136491	-0.027854	0
GDP	0.101620	0.461455	-0.296049	-0.108427	0.354712	0.899373	0.083903	-0.076466	0.301557	-0.112081	0.211976	0.138364	0.200666	-0.136491	1.000000	-0.028270	-0
Population	0.016969	-0.021538	-0.013647	0.556801	-0.035252	-0.025662	-0.123321	0.265966	-0.072301	0.544423	-0.038540	-0.079662	-0.028444	-0.027854	-0.028270	1.000000	0
thinness 1-19 years	-0.047876	-0.477183	0.302904	0.465711	-0.428795	-0.251369	-0.120429	0.224808	-0.532025	0.467789	-0.221823	-0.277101	-0.229518	0.204064	-0.285697	0.253944	1
thinness 5-9 years	-0.050929	-0.471584	0.308457	0.471350	-0.417414	-0.252905	-0.124960	0.221072	-0.538911	0.472263	-0.222592	-0.283774	-0.222743	0.207283	-0.290539	0.251403	0
Income composition of resources	0.243468	0.724776	-0.457626	-0.145139	0.450040	0.381952	0.199549	-0.129568	0.508774	-0.163305	0.381078	0.166682	0.401456	-0.249519	0.460341	-0.008735	-0
Schooling	0.209400	0.751975	-0.454612	-0.193720	0.547378	0.389687	0.231117	-0.137225	0.546961	-0.209373	0.417866	0.246384	0.425332	-0.220429	0.448273	-0.031668	-0

In [21]:
```
plt.figure(figsize=(15,12))
sns.heatmap(corr_df_reg,annot=True)
```

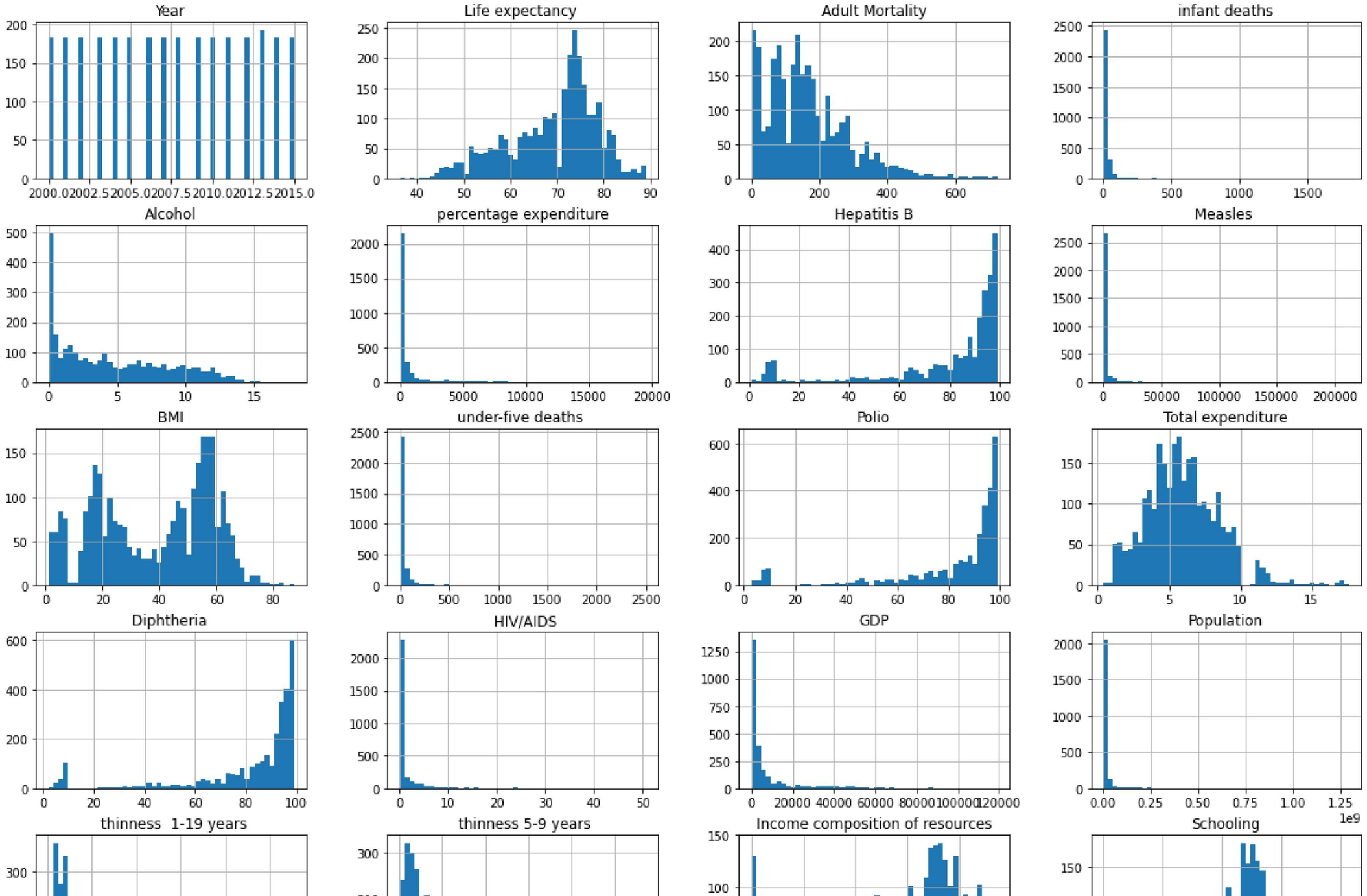
Out[21]: <AxesSubplot:>

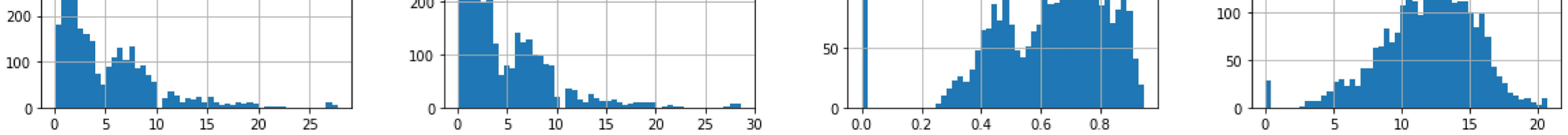


Life e
Adu
in
percentage e
under-
Total e
thinness
thinnes
Income composition o

Have a look visually at the numeric columns.

In [22]:
```
df_reg.hist(bins=50, figsize=(20,15))
plt.show()
```





We have miising vlaues in numeric columns only and hence let us replace those with the mean.

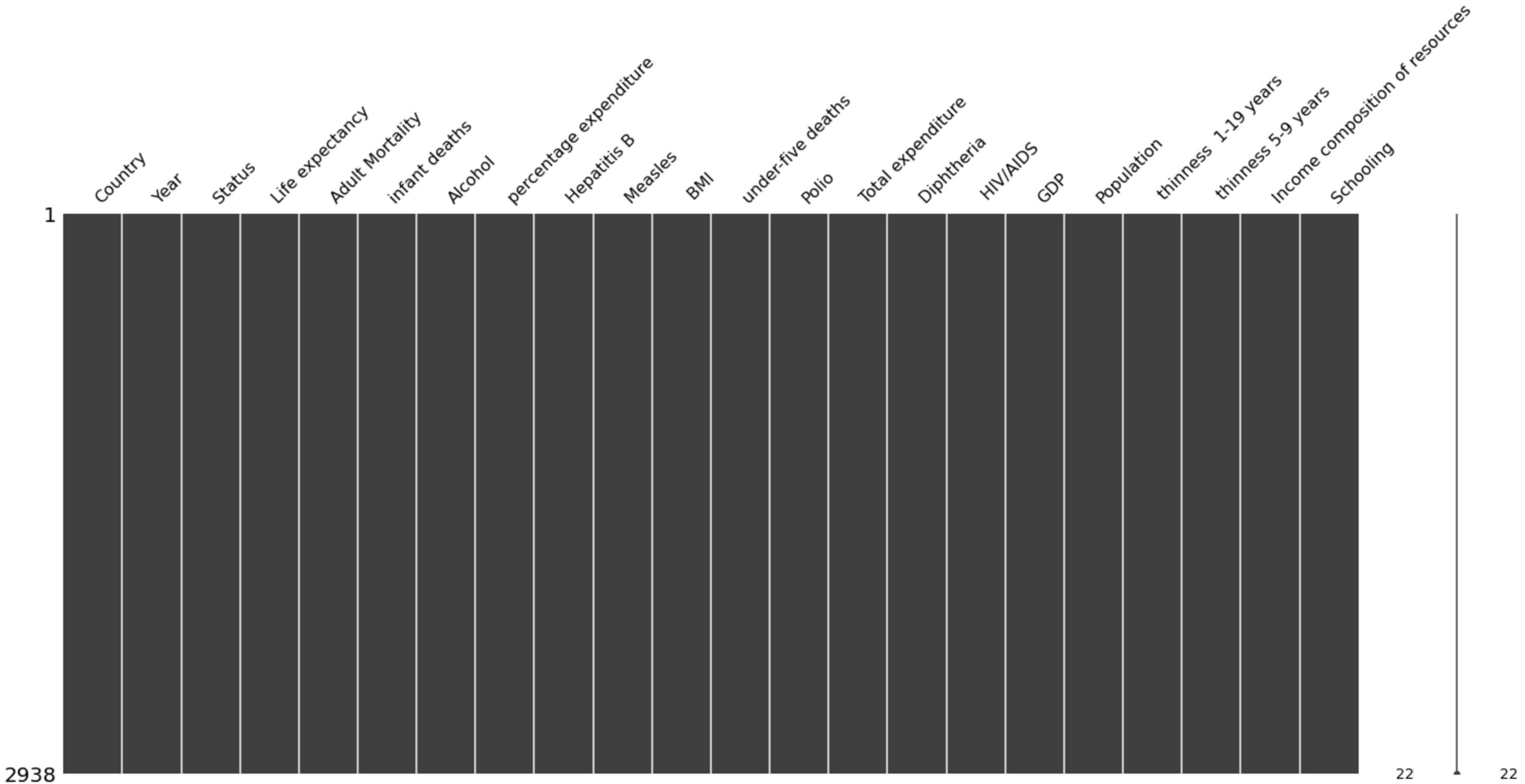
In [24]:
```
def imputer_mean(feature,data=df_reg):
    data[feature].fillna(data[feature].mean(),inplace=True)

features_miss= df_reg.columns[df_reg.isna().any()]
for feature in features_miss:
    imputer_mean(feature=feature)
```

In [25]:
```
df_reg.isnull().sum()
```

Out[25]:
```
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
```

In [26]:
```
msno.matrix(df_reg)
```

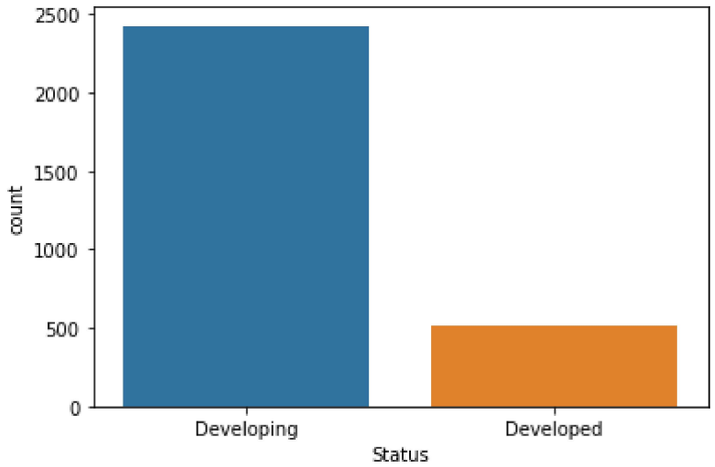
Out[26]: <AxesSubplot:>



Let us explore few aspects of the data before going ahead further.

In [27]:
```
sns.countplot(x='Status',data=df_reg)
```

Out[27]: <AxesSubplot:xlabel='Status', ylabel='count'>



In [28]:
```
df_status = df_reg.groupby('Status')
for sta,life in df_status:
    print(sta + ":" + str(life['Life expectancy '].mean()))
```

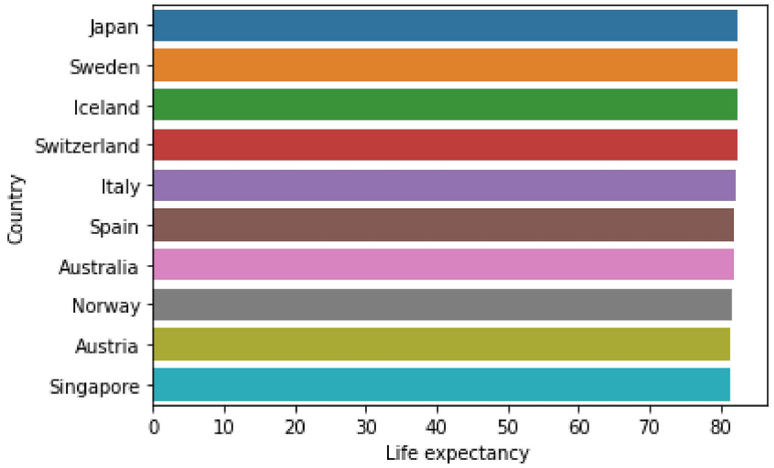
```
Developed:79.19785156249996
Developing:67.12017696493821
```

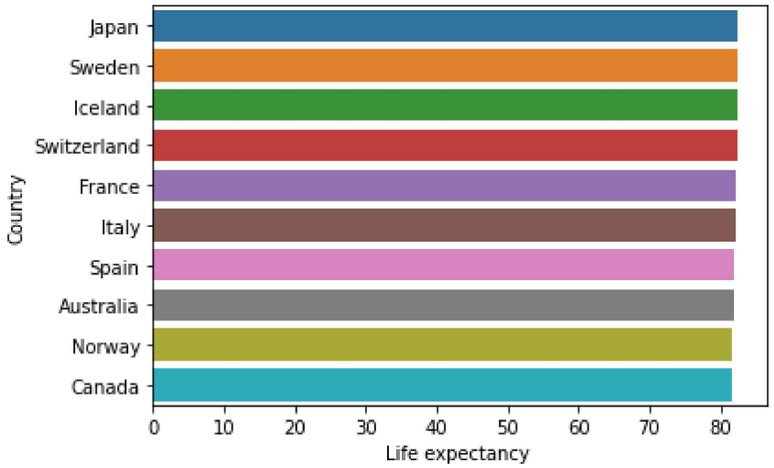
In [29]:
```
print('Top 10 developed countries with the longest life expectancy')
df_sta_ed = df_reg[df_reg.Status=='Developed'].groupby('Country')
print(df_sta_ed['Life expectancy '].mean().sort_values(ascending=False).head(10))
print('#'*50)
print('Top 10 countries with the longest life expectancy')
df_coun = df_reg.groupby('Country')
print(df_coun['Life expectancy '].mean().sort_values(ascending=False).head(10))
print('#'*50)
print('Top 10 countries with the shortest life expectancy')
print(df_coun['Life expectancy '].mean().sort_values(ascending=False).tail(10))
print('#'*50)
```

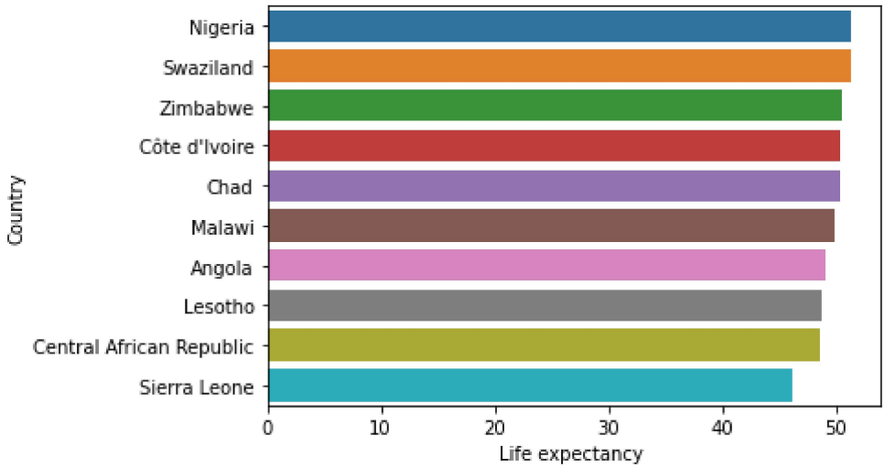
```
Top 10 developed countries with the longest life expectancy
Country
Japan          82.53750
Sweden         82.51875
Iceland        82.44375
Switzerland    82.33125
Italy          82.18750
Spain          82.06875
Australia      81.81250
Norway         81.79375
Austria        81.48125
Singapore      81.47500
Name: Life expectancy , dtype: float64
##################################################
Top 10 countries with the longest life expectancy
Country
Japan          82.53750
Sweden         82.51875
Iceland        82.44375
Switzerland    82.33125
France         82.21875
Italy          82.18750
Spain          82.06875
Australia      81.81250
Norway         81.79375
Canada         81.68750
Name: Life expectancy , dtype: float64
##################################################
Top 10 countries with the shortest life expectancy
Country
Nigeria                    51.35625
Swaziland                  51.32500
Zimbabwe                   50.48750
Côte d'Ivoire              50.38750
Chad                       50.38750
```

```
Malawi                       49.89375
Angola                       49.01875
Lesotho                      48.78125
Central African Republic     48.51250
Sierra Leone                 46.11250
Name: Life expectancy , dtype: float64
##################################################
```

In [30]:
```
df_ed = df_sta_ed['Life expectancy '].mean().sort_values(ascending=False).head(10)
dic = {'Country':df_ed.index,'Life expectancy ':df_ed.values}
df_ed = pd.DataFrame(dic)
df_long = df_coun['Life expectancy '].mean().sort_values(ascending=False).head(10)
dic = {'Country':df_long.index,'Life expectancy ':df_long.values}
df_long = pd.DataFrame(dic)
df_short = df_coun['Life expectancy '].mean().sort_values(ascending=False).tail(10)
dic = {'Country':df_short.index,'Life expectancy ':df_short.values}
df_short = pd.DataFrame(dic)
for df in [df_ed,df_long,df_short]:
    sns.barplot(x='Life expectancy ',y='Country',data=df)
    plt.show()
```







Let us seperate the lable and features.

In [31]:
```
y_base = df_reg['Life expectancy ']
X_base = df_reg.drop('Life expectancy ',axis=1)
X_base.head()
```

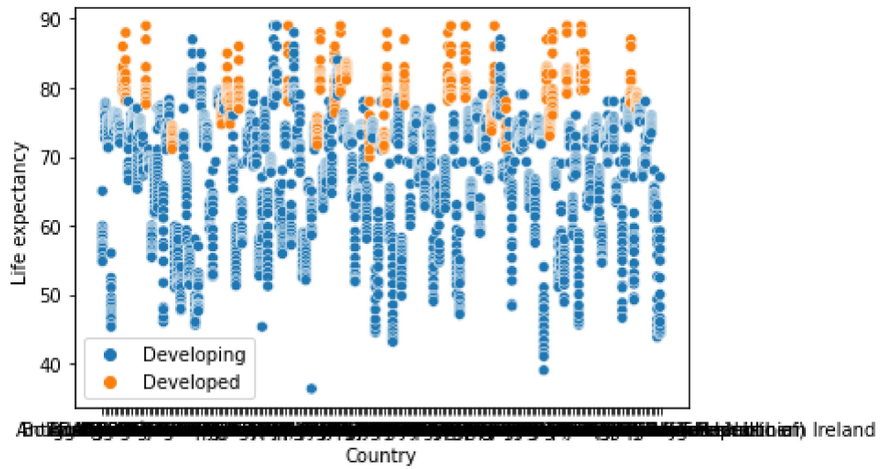
Out[31]:

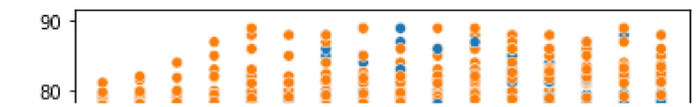
	Country	Year	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Inc compos resou
0	Afghanistan	2015	Developing	263.0	62	0.01	71.279624	65.0	1154	19.1	...	6.0	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3	
1	Afghanistan	2014	Developing	271.0	64	0.01	73.523582	62.0	492	18.6	...	58.0	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5	
2	Afghanistan	2013	Developing	268.0	66	0.01	73.219243	64.0	430	18.1	...	62.0	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7	
3	Afghanistan	2012	Developing	272.0	69	0.01	78.184215	67.0	2787	17.6	...	67.0	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0	
4	Afghanistan	2011	Developing	275.0	71	0.01	7.097109	68.0	3013	17.2	...	68.0	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2	

5 rows × 21 columns

Let us explore the features with respect to label visually.

In [32]:
```
# X.columns
for features in X_base.columns:
    if features == 'Status':
        pass
    else:
        sns.scatterplot(x=X_base[features],y=y_base,hue=X_base['Status'])
        plt.legend()
        plt.show()
```





Perform lable encoding on Country and Year columns

In [33]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X_base['Country'] = le.fit_transform(X_base['Country'])
X_base['Year'] = le.fit_transform(X_base['Year'])
# X.head(3)
# X.tail(3)
```

In [34]:
```
X_base.head()
```

Out[34]:

	Country	Year	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Incom compositio resource
0	0	15	Developing	263.0	62	0.01	71.279624	65.0	1154	19.1	...	6.0	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3	0.47
1	0	14	Developing	271.0	64	0.01	73.523582	62.0	492	18.6	...	58.0	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5	0.47
2	0	13	Developing	268.0	66	0.01	73.219243	64.0	430	18.1	...	62.0	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7	0.47
3	0	12	Developing	272.0	69	0.01	78.184215	67.0	2787	17.6	...	67.0	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0	0.46
4	0	11	Developing	275.0	71	0.01	7.097109	68.0	3013	17.2	...	68.0	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2	0.45

5 rows × 21 columns

Convert the Status categorical feature using dummies

In [35]:
```
X_all = pd.get_dummies(X_base, ['Status'], drop_first=True)
```

In [36]:
```
X_all.head(10)
```

Out[36]:

	Country	Year	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	...	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooli
0	0	15	263.0	62	0.01	71.279624	65.0	1154	19.1	83	...	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3	0.479	1
1	0	14	271.0	64	0.01	73.523582	62.0	492	18.6	86	...	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5	0.476	1
2	0	13	268.0	66	0.01	73.219243	64.0	430	18.1	89	...	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7	0.470	
3	0	12	272.0	69	0.01	78.184215	67.0	2787	17.6	93	...	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0	0.463	
4	0	11	275.0	71	0.01	7.097109	68.0	3013	17.2	97	...	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2	0.454	
5	0	10	279.0	74	0.01	79.679367	66.0	1989	16.7	102	...	9.20	66.0	0.1	553.328940	2883167.0	18.4	18.4	0.448	
6	0	9	281.0	77	0.01	56.762217	63.0	2861	16.2	106	...	9.42	63.0	0.1	445.893298	284331.0	18.6	18.7	0.434	
7	0	8	287.0	80	0.03	25.873925	64.0	1599	15.7	110	...	8.33	64.0	0.1	373.361116	2729431.0	18.8	18.9	0.433	
8	0	7	295.0	82	0.02	10.910156	63.0	1141	15.2	113	...	6.73	63.0	0.1	369.835796	26616792.0	19.0	19.1	0.415	
9	0	6	295.0	84	0.03	17.171518	64.0	1990	14.7	116	...	7.43	58.0	0.1	272.563770	2589345.0	19.2	19.3	0.405	

10 rows × 21 columns

In [37]:
```
X_all['Status_Developing'].value_counts()
```

Out[37]:
```
1    2426
0     512
Name: Status_Developing, dtype: int64
```

Let us apply standard scaler on the features.

In [38]:
```
from sklearn.preprocessing import StandardScaler
scs = StandardScaler()
X_alll = pd.DataFrame(scs.fit_transform(X_all))
```

In [39]:
```
X_alll.head()
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18
0	-1.691042	1.621762	0.791586	0.268824	-1.172958	-0.335570	-0.705861	-0.110384	-0.964715	0.255359	...	0.925806	-0.732952	-0.323445	-0.525248	0.389975	2.813130	2.773279	-0.725401
1	-1.691042	1.404986	0.856072	0.285786	-1.172958	-0.334441	-0.838704	-0.168124	-0.989810	0.274060	...	0.934140	-0.859877	-0.323445	-0.523083	-0.230936	2.881408	2.817902	-0.740050
2	-1.691042	1.188210	0.831890	0.302749	-1.172958	-0.334594	-0.750142	-0.173531	-1.014905	0.292761	...	0.913306	-0.775260	-0.323445	-0.521632	0.352715	2.926927	2.862526	-0.769349
3	-1.691042	0.971434	0.864132	0.328193	-1.172958	-0.332096	-0.617299	0.032045	-1.040000	0.317696	...	1.075815	-0.648335	-0.323445	-0.518723	-0.168315	2.972446	2.929461	-0.803531
4	-1.691042	0.754658	0.888314	0.345155	-1.172958	-0.367862	-0.573018	0.051757	-1.060076	0.342631	...	0.804966	-0.606027	-0.323445	-0.564893	-0.181666	3.040724	2.974085	-0.847480

5 rows × 21 columns

In [40]:
```
y_base.head()
```

Out[40]:
```
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy , dtype: float64
```

Let us use OLS to Check the regression results.

In [41]:
```
import statsmodels.api as sm
X2 = sm.add_constant(X_alll)
ols = sm.OLS(y_base,X2)
lr = ols.fit()
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Life expectancy    R-squared:                       0.820
Model:                           OLS   Adj. R-squared:                  0.819
Method:                Least Squares   F-statistic:                     634.5
Date:               Fri, 28 May 2021   Prob (F-statistic):               0.00
Time:                       01:19:48   Log-Likelihood:                -8262.3
No. Observations:                2938   AIC:                         1.657e+04
Df Residuals:                    2916   BIC:                         1.670e+04
Df Model:                          21
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         69.2249      0.075    928.035      0.000      69.079      69.371
0              0.2555      0.076      3.361      0.001       0.106       0.405
1             -0.0428      0.080     -0.536      0.592      -0.200       0.114
2             -2.4625      0.099    -24.981      0.000      -2.656      -2.269
3             11.9566      0.994     12.034      0.000      10.008      13.905
4              0.2785      0.102      2.720      0.007       0.078       0.479
5              0.1991      0.168      1.183      0.237      -0.131       0.529
6             -0.3219      0.088     -3.646      0.000      -0.495      -0.149
7             -0.2234      0.088     -2.548      0.011      -0.395      -0.051
8              0.8684      0.098      8.847      0.000       0.676       1.061
9            -12.1661      0.991    -12.282      0.000     -14.108     -10.224
10             0.6520      0.104      6.257      0.000       0.448       0.856
11             0.1424      0.082      1.732      0.083      -0.019       0.304
12             0.9519      0.111      8.568      0.000       0.734       1.170
13            -2.4151      0.090    -26.900      0.000      -2.591      -2.239
14             0.4199      0.171      2.459      0.014       0.085       0.755
15             0.0104      0.091      0.115      0.909      -0.168       0.189
16            -0.3268      0.221     -1.478      0.140      -0.761       0.107
17            -0.0092      0.222     -0.042      0.967      -0.445       0.427
18             1.1824      0.131      9.019      0.000       0.925       1.439
19             2.1510      0.136     15.759      0.000       1.883       2.419
20            -0.5715      0.103     -5.565      0.000      -0.773      -0.370
==============================================================================
Omnibus:                      141.341   Durbin-Watson:                   0.703
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              427.781
Skew:                          -0.176   Prob(JB):                     1.28e-93
Kurtosis:                       4.836   Cond. No.                         45.6
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Let us using backward elimination to drop the feature giving p-value less than 0.05

In [42]:
```
while (lr.pvalues.max()>0.05):
    X2.drop(lr.pvalues.idxmax(),axis=1,inplace=True)
    ols = sm.OLS(y_base,X2)
    lr = ols.fit()
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Life expectancy    R-squared:                       0.820
Model:                           OLS   Adj. R-squared:                  0.819
Method:                Least Squares   F-statistic:                     832.4
Date:               Fri, 28 May 2021   Prob (F-statistic):               0.00
Time:                       01:20:45   Log-Likelihood:                -8264.9
No. Observations:                2938   AIC:                         1.656e+04
Df Residuals:                    2921   BIC:                         1.667e+04
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         69.2249      0.075    928.006      0.000      69.079      69.371
0              0.2577      0.076      3.404      0.001       0.109       0.406
2             -2.4723      0.098    -25.177      0.000      -2.665      -2.280
3             12.0236      0.976     12.323      0.000      10.110      13.937
4              0.3144      0.101      3.124      0.002       0.117       0.512
6             -0.3355      0.088     -3.819      0.000      -0.508      -0.163
7             -0.2286      0.087     -2.616      0.009      -0.400      -0.057
8              0.8770      0.097      9.048      0.000       0.687       1.067
9            -12.2259      0.981    -12.461      0.000     -14.150     -10.302
10             0.6503      0.104      6.248      0.000       0.446       0.854
12             0.9608      0.111      8.673      0.000       0.744       1.178
13            -2.3953      0.089    -26.927      0.000      -2.570      -2.221
14             0.5863      0.088      6.670      0.000       0.414       0.759
16            -0.3558      0.104     -3.405      0.001      -0.561      -0.151
18             1.1442      0.129      8.855      0.000       0.891       1.398
19             2.1578      0.136     15.887      0.000       1.891       2.424
20            -0.6119      0.101     -6.057      0.000      -0.810      -0.414
==============================================================================
Omnibus:                      134.011   Durbin-Watson:                   0.700
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              393.905
Skew:                          -0.167   Prob(JB):                     2.91e-86
Kurtosis:                       4.763   Cond. No.                         41.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [43]:
```
X2.head()
```

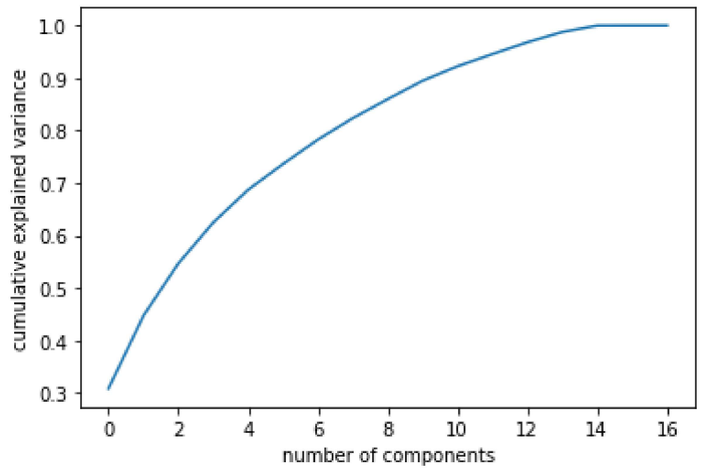
Out[43]:

	const	0	2	3	4	6	7	8	9	10	12	13	14	16	18	19	20
0	1.0	-1.691042	0.791586	0.268824	-1.172958	-0.705861	-0.110384	-0.964715	0.255359	-3.278638	-0.732952	-0.323445	-0.525248	2.813130	-0.725401	-0.579931	0.459399
1	1.0	-1.691042	0.856072	0.285786	-1.172958	-0.838704	-0.168124	-0.989810	0.274060	-1.051482	-0.859877	-0.323445	-0.523083	2.881408	-0.740050	-0.610570	0.459399
2	1.0	-1.691042	0.831890	0.302749	-1.172958	-0.750142	-0.173531	-1.014905	0.292761	-0.880163	-0.775260	-0.323445	-0.521632	2.926927	-0.769349	-0.641209	0.459399
3	1.0	-1.691042	0.864132	0.328193	-1.172958	-0.617299	0.032045	-1.040000	0.317696	-0.666013	-0.648335	-0.323445	-0.518723	2.972446	-0.803531	-0.671847	0.459399
4	1.0	-1.691042	0.888314	0.345155	-1.172958	-0.573018	0.051757	-1.060076	0.342631	-0.623183	-0.606027	-0.323445	-0.564893	3.040724	-0.847480	-0.763764	0.459399

Let us use PCA to check whether we can use drastically fewer number of principal componetnts or not.

In [44]:
```
from sklearn.decomposition import PCA
pca = PCA().fit(X2)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

Out[44]:
```
Text(0, 0.5, 'cumulative explained variance')
```



But we see that we have to take 14 out of 16 to fully capture the gamut of variance and hence we did not go for using 14 by dropping only two.

Let us drop the const column as it is not part of the database.

In [45]:
```
X_fin=X2.drop('const',axis=1)
y_fin=y_base
X_fin.head()
```

Out[45]:

	0	2	3	4	6	7	8	9	10	12	13	14	16	18	19	20
0	-1.691042	0.791586	0.268824	-1.172958	-0.705861	-0.110384	-0.964715	0.255359	-3.278638	-0.732952	-0.323445	-0.525248	2.813130	-0.725401	-0.579931	0.459399
1	-1.691042	0.856072	0.285786	-1.172958	-0.838704	-0.168124	-0.989810	0.274060	-1.051482	-0.859877	-0.323445	-0.523083	2.881408	-0.740050	-0.610570	0.459399
2	-1.691042	0.831890	0.302749	-1.172958	-0.750142	-0.173531	-1.014905	0.292761	-0.880163	-0.775260	-0.323445	-0.521632	2.926927	-0.769349	-0.641209	0.459399
3	-1.691042	0.864132	0.328193	-1.172958	-0.617299	0.032045	-1.040000	0.317696	-0.666013	-0.648335	-0.323445	-0.518723	2.972446	-0.803531	-0.671847	0.459399
4	-1.691042	0.888314	0.345155	-1.172958	-0.573018	0.051757	-1.060076	0.342631	-0.623183	-0.606027	-0.323445	-0.564893	3.040724	-0.847480	-0.763764	0.459399

In [46]:
```
y_fin.head()
```

Out[46]:
```
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy , dtype: float64
```

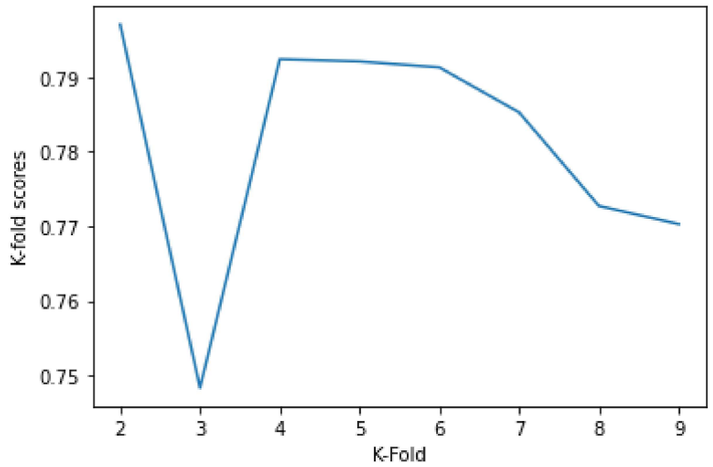
Let us import the required libraries for model development.

In [47]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
cross_val_score(LinearRegression(),X_fin,y_fin,cv=5).mean()
```

Out[47]:
```
0.7920794860943001
```

Try to find the best value for k for from k-fold cross validation.

In [48]:
```
#Graph k-fold score for Linear Regression

l_scores = []
for i in range(2,10):
    l_scores.append(cross_val_score(LinearRegression(),X_fin,y_fin,cv=i).mean())



plt.plot(range(2,10),l_scores)
plt.xlabel('K-Fold')
plt.ylabel('K-fold scores')
plt.show()
```



From the above K=4 seems to be the optimum one.

Now, let us perform the test, train split from lable and features.

In [49]:
```
from sklearn.model_selection import train_test_split
X_train_lr,X_test_lr,y_train_lr,y_test_lr = train_test_split(X_fin,y_fin,
                                                    random_state=1,test_size=0.20)
```

Let us build the model.

In [50]:
```
# Let us build the model using using X_train and y_train and check the score using X_test and y_test
model_lr = LinearRegression()
model_lr.fit(X_train_lr,y_train_lr)
model_lr.score(X_test_lr,y_test_lr)
```

Out[50]:
```
0.7954100686387191
```

Let us now perform the evaluation of the model's performance.

In [51]:
```
y_pred_lr = model_lr.predict(X_test_lr)
from sklearn.metrics import r2_score,mean_squared_error
import math

print(f'R2 Score: {r2_score(y_test_lr,y_pred_lr)}')
print(f'Mean Squared Error, MSE: {mean_squared_error(y_test_lr,y_pred_lr)}')
print(f'RMSE: {math.sqrt(mean_squared_error(y_test_lr,y_pred_lr))}')
```

```
R2 Score: 0.7954100686387191
Mean Squared Error, MSE: 17.67405847480355
RMSE: 4.204052625122996
```

In [52]:
```
y_test_lr.head()
```

Out[52]:
```
1268    82.1
2719    58.4
2710    63.3
2028    67.9
351     47.8
Name: Life expectancy , dtype: float64
```

In [53]:
```
y_pred_lr[:5]
```

Out[53]:
```
array([79.30509709, 57.09871637, 65.13334181, 68.26389288, 40.5149948 ])
```

In [54]:
```
# Let us find the number of observations and number of features to be used in calculation for the adjusted R2
nl = len(X_test_lr)
kl = len(X_test_lr.iloc[0])
R2l = r2_score(y_test_lr,y_pred_lr)
R2l
```

Out[54]:
```
0.7954100686387191
```

In [55]:
```
Adj_R2l = 1 - ((nl-1)*(1- R2l)/(nl-kl-1))
print(f'The adjusted R2: {Adj_R2l}')
```

```
The adjusted R2: 0.7896772509473347
```

The performance of the model seems to be a goodish one from the value of value of R2 and Adjusted R2.

```
################################### K Nearest Neighbors  ##############################################
```

Let us import the requied libraries for the KNN model

In [56]:
```
#import the knn model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
knn = KNeighborsRegressor()
```

Let us take the processed fetures and label from the linear regression pre-processing.

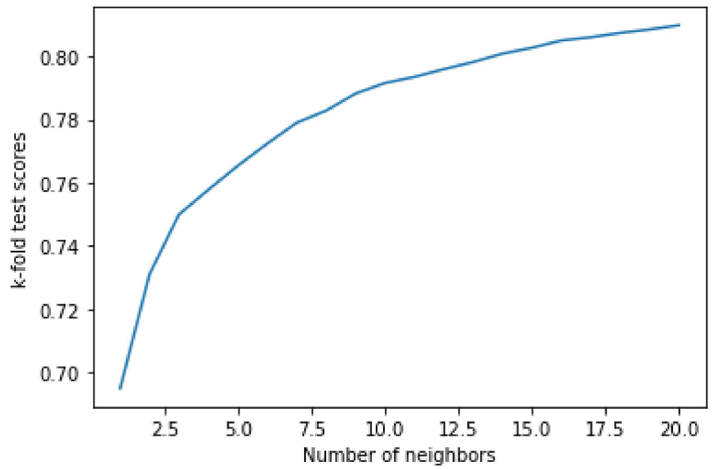
In [57]:
```
X_knn=X_fin
y_knn=y_fin
```

Performing scaling of the features using standard scaling

In [58]:
```
sc = StandardScaler()
X_knn = pd.DataFrame(sc.fit_transform(X_knn))
y_knn = pd.Series(y_knn)
```

In [59]:
```
#for no.of neighbors from 1 - 20, graph the k-fold scores

scores = []
for i in range(1,21,1):
    knn = KNeighborsRegressor(n_neighbors=i, weights='distance',p=2)
    scores.append(cross_val_score(knn,X_knn,y_knn,cv=5).mean())
```

In [60]:
```
import matplotlib.pyplot as plt
plt.plot(range(1,21,1),scores)
plt.xlabel('Number of neighbors')
plt.ylabel('k-fold test scores')
plt.show()
```



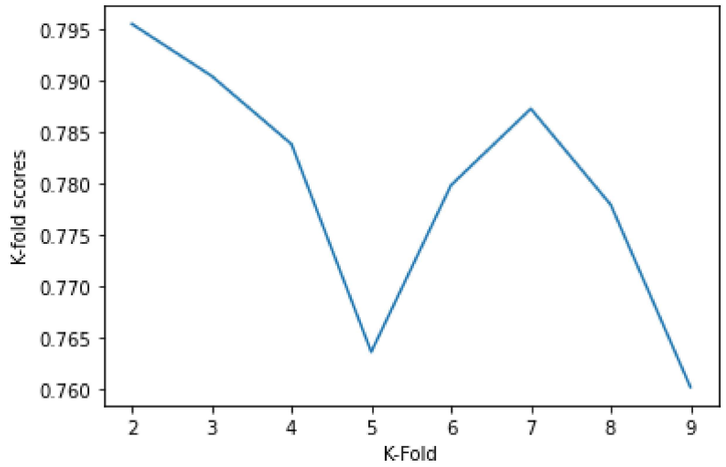
In [61]:
```
X_knn.shape
```

Out[61]: (2938, 16)

In [62]:
```
y_knn.shape
```

Out[62]: (2938,)

In [63]:
```
#Graph k-fold score for KNN

k_scores = []
for i in range(2,10):
    k_scores.append(cross_val_score(KNeighborsRegressor(),X_knn,y_knn,cv=i).mean())

plt.plot(range(2,10),k_scores)
plt.xlabel('K-Fold')
plt.ylabel('K-fold scores')
plt.show()
```



Here, k=7 seems to be the optimum one.

In [64]:
```
from sklearn.model_selection import train_test_split
X_train_k,X_test_k,y_train_k,y_test_k = train_test_split(X_knn,y_knn,
                                                         random_state=1)
```

In [65]:
```
# Let us build the model using using X_train and y_train and check the score using X_test and y_test
model_knn = KNeighborsRegressor()
model_knn.fit(X_train_k,y_train_k)
model_knn.score(X_test_k,y_test_k)
```

Out[65]: 0.8962654820785558

In [66]:
```
y_pred_k = model_knn.predict(X_test_k)
from sklearn.metrics import r2_score,mean_squared_error
import math

print(f'R2 Score: {r2_score(y_test_k,y_pred_k)}')
print(f'Mean Squared Error, MSE: {mean_squared_error(y_test_k,y_pred_k)}')
print(f'RMSE: {math.sqrt(mean_squared_error(y_test_k,y_pred_k))}')
```

```
R2 Score: 0.8962654820785558
Mean Squared Error, MSE: 8.838014022178921
RMSE: 2.9728797523914285
```

In [67]:
```
# Let us find the number of observations and number of features to be used in calculation for the adjusted R2
nk = len(X_test_k)
kk = len(X_test_k.iloc[0])
R2k = r2_score(y_test_k,y_pred_k)
R2k
```

Out[67]:
```
0.8962654820785558
```

In [68]:
```
Adj_R2k = 1 - ((nk-1)*(1- R2k)/(nk-kk-1))
print(f'The adjusted R2: {Adj_R2k}')
```

```
The adjusted R2: 0.8939538493672144
```

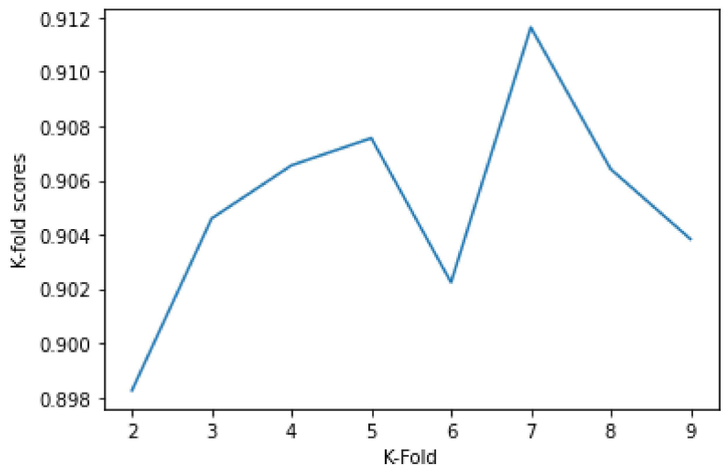
```
The performance of the model is much better as compared to the linear regression one.
```

```
########################################### Random Forest ##############################################
```

In [69]:
```
#import the Random Forest model
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
rfr = RandomForestRegressor()
```

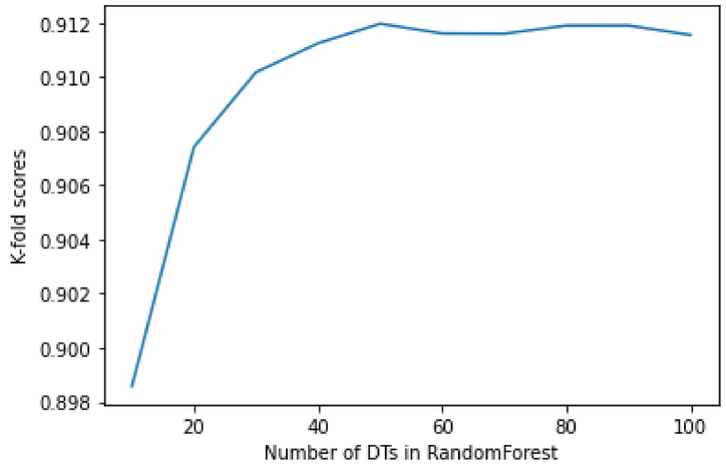
In [70]:
```
X_r=X_fin
y_r=y_fin
```

In [71]:
```
#Graph k-fold score for Random Forest

r_scores = []
for i in range(2,10):
    r_scores.append(cross_val_score(RandomForestRegressor(),X_r,y_r,cv=i).mean())



plt.plot(range(2,10),r_scores)
plt.xlabel('K-Fold')
plt.ylabel('K-fold scores')
plt.show()
```



In [72]:
```
#Graph k-fold score vs no. of estimators in Random Forest
rf_scores = []
for i in range(10,101,10):
    rf_scores.append(cross_val_score(RandomForestRegressor(n_estimators=i,random_state=9),
                                     X_r,y_r,cv=7).mean())
```

In [73]:
```
plt.plot(range(10,101,10),rf_scores)
plt.xlabel('Number of DTs in RandomForest')
plt.ylabel('K-fold scores')
plt.show()
```



Number of DT=50 seems to be optimum one.

Let us perform gridsearch parameter tuning.

In [74]:
```
params = {
            'n_estimators': [90, 100,110,120,130],
            'max_depth': [12,13,14,15]
        }
model_rfr = GridSearchCV(RandomForestRegressor(), params,cv=7)
model_rfr.fit(X_r,y_r)
```

Out[74]:
```
GridSearchCV(cv=7, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [12, 13, 14, 15],
                         'n_estimators': [90, 100, 110, 120, 130]})
```

The best parameter found as given below:

In [75]:
```
model_rfr.best_params_
```

Out[75]:
```
{'max_depth': 13, 'n_estimators': 100}
```

In [76]:
```
model_rfr.best_score_
```

Out[76]:
```
0.9136803665596555
```

Let us now find the model with the best estimator from the parameter tuning.

In [77]:
```
best_model_rfr = model_rfr.best_estimator_
```

In [78]:
```
from sklearn.model_selection import train_test_split
X_train_rfr,X_test_rfr,y_train_rfr,y_test_rfr = train_test_split(X_r,y_r,
                                                random_state=5)
```

In [79]:
```
best_model_rfr.fit(X_train_rfr,y_train_rfr)
```

Out[79]:
```
RandomForestRegressor(max_depth=13)
```

In [80]:
```
best_model_rfr.score(X_test_rfr,y_test_rfr)
```

Out[80]:
```
0.9647003980751055
```

In [81]:
```
cross_val_score(RandomForestRegressor(n_estimators=120,max_depth=12),X_r,y_r,cv=7)
```

Out[81]:
```
array([0.91957785, 0.93719439, 0.88039985, 0.93699538, 0.92395707,
       0.87805821, 0.91235222])
```

In [82]:
```
y_pred_rfr = best_model_rfr.predict(X_test_rfr)
from sklearn.metrics import r2_score,mean_squared_error
import math

print(f'R2 Score: {r2_score(y_test_rfr,y_pred_rfr)}')
print(f'Mean Squared Error, MSE: {mean_squared_error(y_test_rfr,y_pred_rfr)}')
print(f'RMSE: {math.sqrt(mean_squared_error(y_test_rfr,y_pred_rfr))}')
```

```
R2 Score: 0.9647003980751055
Mean Squared Error, MSE: 3.2079059188199817
RMSE: 1.7910627903063538
```

In [83]:
```
# Let us find the number of observations and number of features to be used in calculation for the adjusted R2
nr = len(X_test_rfr)
kr = len(X_test_rfr.iloc[0])
R2r = r2_score(y_test_rfr,y_pred_rfr)
R2r
```

Out[83]:
```
0.9647003980751055
```

In [84]:
```
Adj_R2r = 1 - ((nr-1)*(1- R2r)/(nr-kr-1))
print(f'The adjusted R2: {Adj_R2r}')
```

```
The adjusted R2: 0.9639137774193975
```

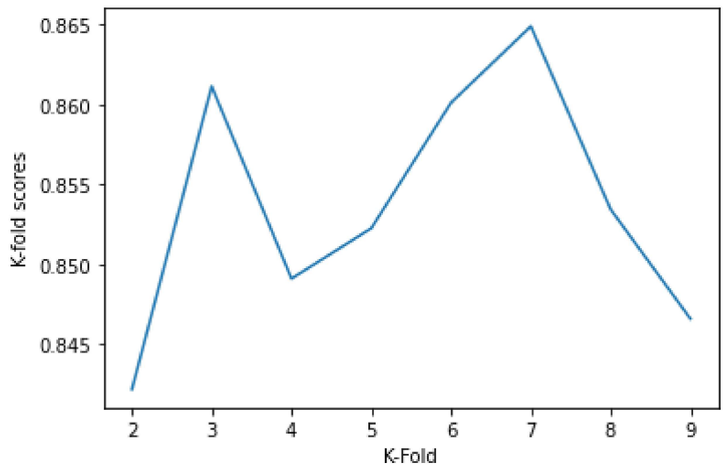
```
The performance of this model is very good as we find from the R2 and Adjusted R2
```

```
################################################ Adaboost ################################################
```

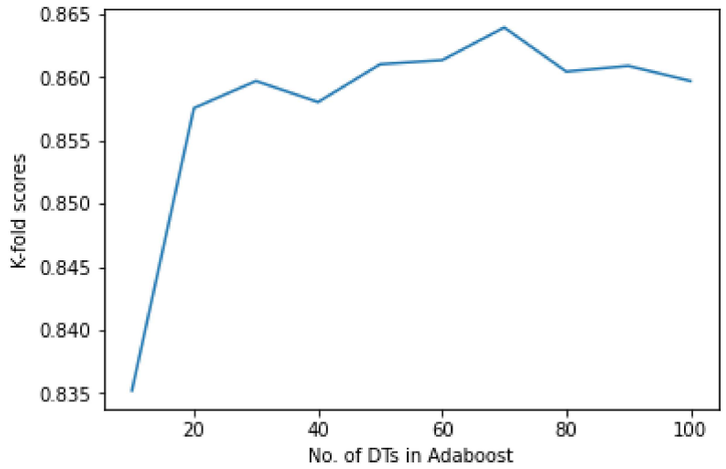
In [85]:
```
#import the adaboost model
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
adar = RandomForestRegressor()
```

In [86]:
```
X_a=X_fin
y_a=y_fin
```

In [87]:
```
#Graph k-fold score for AdaBoost

a_scores = []
for i in range(2,10):
    a_scores.append(cross_val_score(AdaBoostRegressor(),X_a,y_a,cv=i).mean())



plt.plot(range(2,10),a_scores)
plt.xlabel('K-Fold')
plt.ylabel('K-fold scores')
plt.show()
```



In [88]:
```
#Graph k-fold score vs no. of estimators in Adaboost which uses DT as base estimators
ada_scores = []
for i in range(10,101,10):
    ada_scores.append(cross_val_score(AdaBoostRegressor(n_estimators=i,random_state=0),
                                      X_a,y_a,cv=7).mean())
plt.plot(range(10,101,10),ada_scores)
plt.xlabel('No. of DTs in Adaboost')
plt.ylabel('K-fold scores')
plt.show()
```



In [89]:
```
from sklearn.tree import DecisionTreeRegressor
#including other params like max_depth, we will apply gridsearch to fine the best settings
params = {
            'n_estimators': [70,80,90,100],
            'base_estimator': [DecisionTreeRegressor(max_depth=9,random_state=0),
                               DecisionTreeRegressor(max_depth=10,random_state=0),
                               DecisionTreeRegressor(max_depth=11,random_state=0)]
        }
model_adar = GridSearchCV(AdaBoostRegressor(random_state=0), params,cv=7)
model_adar.fit(X_a,y_a)
```

Out[89]:
```
GridSearchCV(cv=7, estimator=AdaBoostRegressor(random_state=0),
             param_grid={'base_estimator': [DecisionTreeRegressor(max_depth=9,
                                                                  random_state=0),
                                            DecisionTreeRegressor(max_depth=10,
                                                                  random_state=0),
                                            DecisionTreeRegressor(max_depth=11,
                                                                  random_state=0)],
                         'n_estimators': [70, 80, 90, 100]})
```

In [90]:
```
model_adar.best_params_
```

Out[90]:
```
{'base_estimator': DecisionTreeRegressor(max_depth=10, random_state=0),
 'n_estimators': 80}
```

In [91]:
```
model_adar.best_score_
```

Out[91]:
```
0.9125502312572363
```

In [92]:
```
best_model_adar = model_adar.best_estimator_
```

In [93]:
```
from sklearn.model_selection import train_test_split
X_train_adar,X_test_adar,y_train_adar,y_test_adar = train_test_split(X_a,y_a,
                                                    random_state=0)
```

In [94]:
```
best_model_adar.fit(X_train_adar,y_train_adar)
```

Out[94]:
```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=10,
                                                       random_state=0),
                  n_estimators=80, random_state=0)
```

In [95]:
```
best_model_adar.score(X_test_adar,y_test_adar)
```

Out[95]:
```
0.9623913309393325
```

In [96]:
```
y_pred_adar = best_model_adar.predict(X_test_adar)
from sklearn.metrics import r2_score,mean_squared_error
import math

print(f'R2 Score: {r2_score(y_test_adar,y_pred_adar)}')
print(f'Mean Squared Error, MSE: {mean_squared_error(y_test_adar,y_pred_adar)}')
print(f'RMSE: {math.sqrt(mean_squared_error(y_test_adar,y_pred_adar))}')
```

```
R2 Score: 0.9623913309393325
Mean Squared Error, MSE: 3.524677056861552
RMSE: 1.8774123300067973
```

In [97]:
```
# Let us find the number of observations and number of features to be used in calculation for the adjusted R2
na = len(X_test_adar)
ka = len(X_test_adar.iloc[0])
R2a = r2_score(y_test_adar,y_pred_adar)
R2a
```

Out[97]: 0.9623913309393325

In [98]:
```
Adj_R2a = 1 - ((na-1)*(1- R2a)/(na-ka-1))
print(f'The adjusted R2: {Adj_R2a}')
```

```
The adjusted R2: 0.9615532547485656
```

The performance of this model is also very good and similar to that of Random Forest Model

SVR

In [99]:
```
#import the SVR model
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
svrr = SVR()
```

In [100]:
```
X_svr=X_fin
y_svr=y_fin
```

In [101]:
```
X_svr.head()
```

Out[101]:

	0	2	3	4	6	7	8	9	10	12	13	14	16	18	19	20
0	-1.691042	0.791586	0.268824	-1.172958	-0.705861	-0.110384	-0.964715	0.255359	-3.278638	-0.732952	-0.323445	-0.525248	2.813130	-0.725401	-0.579931	0.459399
1	-1.691042	0.856072	0.285786	-1.172958	-0.838704	-0.168124	-0.989810	0.274060	-1.051482	-0.859877	-0.323445	-0.523083	2.881408	-0.740050	-0.610570	0.459399
2	-1.691042	0.831890	0.302749	-1.172958	-0.750142	-0.173531	-1.014905	0.292761	-0.880163	-0.775260	-0.323445	-0.521632	2.926927	-0.769349	-0.641209	0.459399
3	-1.691042	0.864132	0.328193	-1.172958	-0.617299	0.032045	-1.040000	0.317696	-0.666013	-0.648335	-0.323445	-0.518723	2.972446	-0.803531	-0.671847	0.459399
4	-1.691042	0.888314	0.345155	-1.172958	-0.573018	0.051757	-1.060076	0.342631	-0.623183	-0.606027	-0.323445	-0.564893	3.040724	-0.847480	-0.763764	0.459399

In [102]:
```
y_svr.head()
```

Out[102]:
```
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy , dtype: float64
```

In [103]:
```
sc = StandardScaler()
sc = StandardScaler()
X_svr = pd.DataFrame(sc.fit_transform(X_svr))
y_svr = pd.Series(y_svr)
```

In [104]:
```
X_svr.head()
```

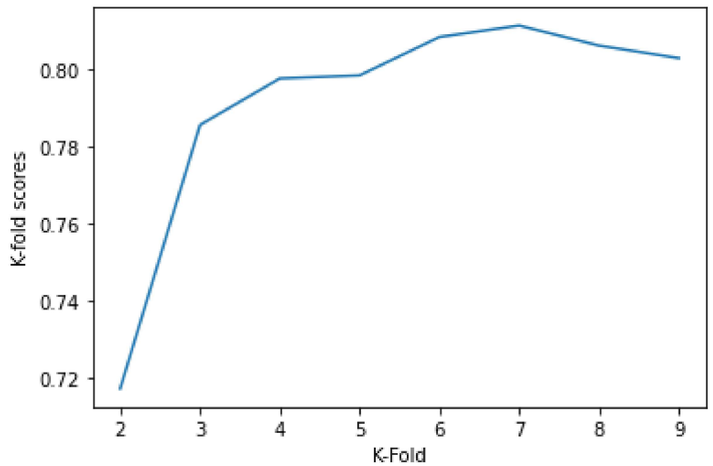
Out[104]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1.691042	0.791586	0.268824	-1.172958	-0.705861	-0.110384	-0.964715	0.255359	-3.278638	-0.732952	-0.323445	-0.525248	2.813130	-0.725401	-0.579931	0.459399
1	-1.691042	0.856072	0.285786	-1.172958	-0.838704	-0.168124	-0.989810	0.274060	-1.051482	-0.859877	-0.323445	-0.523083	2.881408	-0.740050	-0.610570	0.459399
2	-1.691042	0.831890	0.302749	-1.172958	-0.750142	-0.173531	-1.014905	0.292761	-0.880163	-0.775260	-0.323445	-0.521632	2.926927	-0.769349	-0.641209	0.459399
3	-1.691042	0.864132	0.328193	-1.172958	-0.617299	0.032045	-1.040000	0.317696	-0.666013	-0.648335	-0.323445	-0.518723	2.972446	-0.803531	-0.671847	0.459399
4	-1.691042	0.888314	0.345155	-1.172958	-0.573018	0.051757	-1.060076	0.342631	-0.623183	-0.606027	-0.323445	-0.564893	3.040724	-0.847480	-0.763764	0.459399

In [105]:
```
y_svr.head()
```

Out[105]:
```
0    65.0
1    59.9
2    59.9
3    59.5
4    59.2
Name: Life expectancy , dtype: float64
```

In [106]:
```
#Graph k-fold score for SVR

scores = []
for i in range(2,10):
    scores.append(cross_val_score(SVR(),X_svr,y_svr,cv=i).mean())

plt.plot(range(2,10),scores)
plt.xlabel('K-Fold')
plt.ylabel('K-fold scores')
plt.show()
```



In [107]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
X_train_svr, X_test_svr, y_train_svr, y_test_svr = train_test_split(X_svr,y_svr,random_state=0)
```

In [108]:
```
params = {
            'kernel': ('linear', 'rbf'),
            'C': [1, 1.5, 2]
        }
grid_model_svr = GridSearchCV(SVR(gamma='scale',verbose=True),
                          params,cv=7)
grid_model_svr.fit(X_train_svr,y_train_svr)
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Lib
SVM][LibSVM][LibSVM][LibSVM]
```

Out[108]:
```
GridSearchCV(cv=7, estimator=SVR(verbose=True),
             param_grid={'C': [1, 1.5, 2], 'kernel': ('linear', 'rbf')})
```

In [109]:
```
grid_model_svr.best_params_
```

Out[109]:
```
{'C': 2, 'kernel': 'rbf'}
```

In [110]:
```
grid_model_svr.best_score_
```

Out[110]:
```
0.8805991607340529
```

In [111]:
```
model_svr = grid_model_svr.best_estimator_
```

In [112]:
```
model_svr.fit(X_train_svr,y_train_svr)
```

```
[LibSVM]
```

Out[112]:
```
SVR(C=2, verbose=True)
```

In [113]:
```
model_svr.score(X_test_svr,y_test_svr)
```

Out[113]:
```
0.8853626474475291
```

In [114]:
```
y_pred_svr = model_svr.predict(X_test_svr)
from sklearn.metrics import r2_score,mean_squared_error
import math

print(f'R2 Score: {r2_score(y_test_svr,y_pred_svr)}')
print(f'Mean Squared Error, MSE: {mean_squared_error(y_test_svr,y_pred_svr)}')
print(f'RMSE: {math.sqrt(mean_squared_error(y_test_svr,y_pred_svr))}')
```

```
R2 Score: 0.8853626474475291
Mean Squared Error, MSE: 10.743790101937503
RMSE: 3.2777721247727856
```

In [115]:
```
# Let us find the number of observations and number of features to be used in calculation for the adjusted R2
ns = len(X_test_svr)
ks = len(X_test_svr.iloc[0])
R2s = r2_score(y_test_svr,y_pred_svr)
R2s
```

Out[115]:
```
0.8853626474475291
```

In [116]:
```
Adj_R2s = 1 - ((ns-1)*(1- R2s)/(ns-ks-1))
print(f'The adjusted R2: {Adj_R2s}')
```

```
The adjusted R2: 0.8828080546329894
```

The performance of this model is better than the linear regression one but did not give the best performance.

Regression Model Summary

Result	Linear Regression	KNN Regression	Random Forest Regression	AdaBoost Regression	SVR Regression
R2	0.7954	0.8963	0.9649	0.9631	0.8854
Adjusted R2	0.7897	0.8939	0.9642	0.9624	0.8828

- Based on the value of R2 and Adjusted R2 Random Forest and AdaBoost performed the best
- Linear Regression performed the worst of the models
- In-depth knowledge of tuning will help the model perform better

```
############################## Random Forest in pyspark in Anaconda Jupyter Notebook ##########################################
```

Regression Model with PySpark in Anaconda Jupyter Notebook on Local Machine

Let us use the preprocessed data from the sklearn models in local pyspark

In [ ]:
```
# For joining X and y and bring back to a .CSV file to be used in pyspark
result = pd.concat([X_all, y_base], axis=1, join='inner')
result.head(10)
result.to_csv('regression_pyspark.csv', index = False)
```

In [118]:
```
import findspark
findspark.init()
findspark.find()
```

Out[118]:
```
'C:\\spark-3.1.1-bin-hadoop2.7'
```

In [119]:
```
from pyspark.sql import SparkSession
import pyspark
```

In [120]:
```
spark= SparkSession.builder.appName("Random Forest Regression").getOrCreate()
```

In [121]:
```
spark
```

Out[121]:

SparkSession - in-memory

SparkContext

Spark UI (http://DinarTriOSEducation:4040)

Version

v3.1.1

Master

local[*]

AppName

Random Forest Regression

In [122]:
```
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.regression import *
from pyspark.ml.evaluation import *
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

In [123]:
```
df = spark.read.csv('regression_pyspark.csv',inferSchema=True, header=True)
```

In [124]:
```
df.limit(10).toPandas()
```

Out[124]:

	Country	Year	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	...	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling	Status_D
0	0	15	263.0	62	0.01	71.279624	65.0	1154	19.1	83	...	65.0	0.1	584.259210	33736494.0	17.2	17.3	0.479	10.1	
1	0	14	271.0	64	0.01	73.523582	62.0	492	18.6	86	...	62.0	0.1	612.696514	327582.0	17.5	17.5	0.476	10.0	
2	0	13	268.0	66	0.01	73.219243	64.0	430	18.1	89	...	64.0	0.1	631.744976	31731688.0	17.7	17.7	0.470	9.9	
3	0	12	272.0	69	0.01	78.184215	67.0	2787	17.6	93	...	67.0	0.1	669.959000	3696958.0	17.9	18.0	0.463	9.8	
4	0	11	275.0	71	0.01	7.097109	68.0	3013	17.2	97	...	68.0	0.1	63.537231	2978599.0	18.2	18.2	0.454	9.5	
5	0	10	279.0	74	0.01	79.679367	66.0	1989	16.7	102	...	66.0	0.1	553.328940	2883167.0	18.4	18.4	0.448	9.2	
6	0	9	281.0	77	0.01	56.762217	63.0	2861	16.2	106	...	63.0	0.1	445.893298	284331.0	18.6	18.7	0.434	8.9	
7	0	8	287.0	80	0.03	25.873925	64.0	1599	15.7	110	...	64.0	0.1	373.361116	2729431.0	18.8	18.9	0.433	8.7	
8	0	7	295.0	82	0.02	10.910156	63.0	1141	15.2	113	...	63.0	0.1	369.835796	26616792.0	19.0	19.1	0.415	8.4	
9	0	6	295.0	84	0.03	17.171518	64.0	1990	14.7	116	...	58.0	0.1	272.563770	2589345.0	19.2	19.3	0.405	8.1	

10 rows × 22 columns

In [125]:
```
df.printSchema()
```

```
root
 |-- Country: integer (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Adult Mortality: double (nullable = true)
 |-- infant deaths: integer (nullable = true)
 |-- Alcohol: double (nullable = true)
 |-- percentage expenditure: double (nullable = true)
 |-- Hepatitis B: double (nullable = true)
 |-- Measles : integer (nullable = true)
 |--  BMI : double (nullable = true)
 |-- under-five deaths : integer (nullable = true)
 |-- Polio: double (nullable = true)
 |-- Total expenditure: double (nullable = true)
 |-- Diphtheria : double (nullable = true)
 |--  HIV/AIDS: double (nullable = true)
 |-- GDP: double (nullable = true)
 |-- Population: double (nullable = true)
 |--  thinness  1-19 years: double (nullable = true)
 |--  thinness 5-9 years: double (nullable = true)
 |-- Income composition of resources: double (nullable = true)
 |-- Schooling: double (nullable = true)
 |-- Status_Developing: integer (nullable = true)
 |-- Life expectancy : double (nullable = true)
```

In [126]:
```
print(df.count())
print(len(df.columns))
```

```
2938
22
```

In [127]:
```
df.columns
```

Out[127]:
```
['Country',
 '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',
 'Income composition of resources',
 'Schooling',
 'Status_Developing',
 'Life expectancy ']
```

In [128]:
```
input_columns = ['Country',
 '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',
 'Income composition of resources',
 'Schooling',
 'Status_Developing']
```

In [129]:
```
dependent_var = 'Life expectancy '
```

In [130]:
```
assembler = VectorAssembler(inputCols=input_columns, outputCol="features")
feature_vec=assembler.transform(df).select('features',dependent_var)
feature_vec.show(5)
```

```
+--------------------+----------------+
|            features|Life expectancy |
+--------------------+----------------+
|[0.0,15.0,263.0,6...|            65.0|
|[0.0,14.0,271.0,6...|            59.9|
|[0.0,13.0,268.0,6...|            59.9|
|[0.0,12.0,272.0,6...|            59.5|
|[0.0,11.0,275.0,7...|            59.2|
+--------------------+----------------+
only showing top 5 rows
```

In [131]:
```
# Split the data into train and test sets
train_data, test_data = feature_vec.randomSplit([.80,.20],seed=0)
```

In [132]:
```
from pyspark.ml.regression import RandomForestRegressor
r_model = RandomForestRegressor(labelCol=dependent_var, featuresCol="features",
                        maxDepth=15, minInfoGain=0.001, seed=0, numTrees=110)
rfModel = r_model.fit(train_data)

#Evaulation of the Model
predictions = rfModel.transform(test_data)

from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol=dependent_var,metricName='r2')
evaluator.evaluate(predictions)
```

Out[132]:
```
0.9628033866425842
```

In [133]:
```
evaluator_RMSE = RegressionEvaluator(labelCol=dependent_var,metricName='rmse')
evaluator_RMSE.evaluate(predictions)
```

Out[133]:
```
1.7772239064993989
```

In [134]:
```
evaluator_MSE = RegressionEvaluator(labelCol=dependent_var,metricName='mse')
evaluator_MSE.evaluate(predictions)
```

Out[134]:
```
3.1585248138329844
```

In [135]:
```
evaluator_MAE = RegressionEvaluator(labelCol=dependent_var,metricName='mae')
evaluator_MAE.evaluate(predictions)
```

Out[135]:
```
1.1046760378235159
```

In [136]:
```
#Grid Search
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
g_model = RandomForestRegressor(labelCol=dependent_var, featuresCol="features",
                        minInfoGain=0.001, seed=0)
paramGrid = (ParamGridBuilder()\
             .addGrid(g_model.maxDepth,[14,15,16])\
             .addGrid(g_model.numTrees,[100,110,120])\
             .build())

# Create 4-fold CrossValidator
cv = CrossValidator(estimator=g_model, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=4)

cvModel = cv.fit(train_data)
```

In [137]:
```
#Get the best model
rf_bestModel = cvModel.bestModel
```

In [138]:
```
# Feature Importance
# Estimate of the importance of each feature
# Each feature's importance is the average of its importance across all trees
# In the ensemble the importance vector is normalized to sum up to 1.
print(" ")
print('\033[1m' + "Feature Importance"+ '\033[0m')
print("(Scores Up to 1)")
print("Lowest score signifies the least importance")
print(" ")
RF_FeatureImportance = rf_bestModel.featureImportances.toArray()
#Convert from numpy array to list
important_scores = []
for x in RF_FeatureImportance:
    important_scores.append(float(x))

# Then zip with input_columns list and create a df
result_is = spark.createDataFrame(zip(input_columns, important_scores), schema=['feature', 'score'])
print(result_is.orderBy(result_is["score"].desc()).show(truncate=False))

# Make predictions
# PySpark will automatically use the best model when we call fitmodel
predictions_fn = cvModel.transform(test_data)

# Then let us apply it

r2_rf_pys = evaluator.evaluate(predictions_fn)
print(r2_rf_pys)
```

Feature Importance
(Scores Up to 1)
Lowest score signifies the least importance

```
+------------------------------+--------------------+
|feature                       |score               |
+------------------------------+--------------------+
| HIV/AIDS                     |0.31072351545263877 |
|Adult Mortality               |0.19892861396192865 |
|Income composition of resources|0.16572465387890006 |
|Schooling                     |0.08072276350591182 |
| BMI                          |0.03859783444909763 |
|under-five deaths             |0.0291866077025837  |
|Diphtheria                    |0.025209013825530778|
|Polio                         |0.023700940148087057|
|infant deaths                 |0.02138439655182234 |
| thinness 5-9 years           |0.020566885767165647|
| thinness  1-19 years         |0.016925826918868316|
```

```
|Alcohol                        |0.010878319488922898|
|Year                           |0.010375815988753036|
|Status_Developing              |0.009361395062098113|
|Country                        |0.007592708264506554|
|Total expenditure              |0.006770998116881692|
|GDP                            |0.006737716095538603|
|percentage expenditure         |0.004887330857730387|
|Measles                        |0.00421273388375105 |
|Hepatitis B                    |0.003891945736207169|
+-------------------------------+--------------------+
only showing top 20 rows

None
0.9628680228132476
```

The performance of the the Regression model with PySpark is almost same to that of the sklearn

In [ ]: