Life Expectancy Prediction Using ML

By Kaneez Ayesha
Associate – Data Scientist
18-03-2022

Table of Contents

Columns Names and Detail: 3
Libraries: 3
Loading Data: 4
Statistics: 4
Shape of Data: 4
Data Info: 4
Data Cleaning: 5
EDA With Visualization: 6
Correlation: 11
Separating Dependent and Independent Variables: 12
Label Encoder: 12
Min Max Scaler: 13
Train Test Split: 14
Linear regression: 14
Bagging Regressor: 15
XG Boost Predictions with Full data: 15
XG Boost Regression with Train and Test data: 17
Conclusion: 20

Life Expectancy Prediction Using ML

I picked up a dataset from Kaggle website name as "Life expectancy". Life expectancy dataset has 2938 records and 22 columns. The insights from this analysis can be used by Government and Healthcare sectors for the betterment of society. And prediction system to evaluate life expectancy of each and every country of world.

Columns Names and Detail:

Country: Country
Year: Year
Status: Developed or Developing status
Life Expectancy: Life Expectancy in age
Adult Mortality: Adult Mortality Rates of both sexes (probability of dying between 15 and 60 years per 1000 population)
Infant Deaths: Number of Infant Deaths per 1000 population
Alcohol: Alcohol, recorded per capita (15+) consumption (in liters of pure alcohol)
Percentage Expenditure: Expenditure on health as a percentage of Gross Domestic Product per capita (%)
Hepatitis B: Hepatitis B (Hep B) immunization coverage among 1-year-olds (%)
Measles: Measles - number of reported cases per 1000 population.
BMI: Body Mass Index.
Under Five Deaths: deaths per 1,000 live births.
Polio: Between 2 and 10 out of 100 people who have paralysis from poliovirus infection.
Total Expenditure: Health financing is reported as the annual per capita health expenditure.
Diphtheria: Diphtheria
HIV/AIDS:
GDP: Gross Domestic Product, GDP per capita increases by 1%, life expectancy increases by 5.38 years.
Population: Population
Thinness_1_to_19_years:
Thinness_5_to_9_years:
Income Composition of resources: Income composition of resources have the highest correlation coefficient of 0.91 which means that if a country utilizes its resources productively, it is more likely to see its citizens live longer than expected.
Schooling: average based on participation in different levels of education, the expected number of years of schooling may be pulled down by the magnitude of children who never go to school. Those children who are in school may benefit from many more years of education than the average.

Libraries:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import f1_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

Loading Data:

#Reading csv files

df = pd.read_csv("Life Expectancy Data.csv")

df

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1	584.2
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1	612.6
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1	631.7
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1	669.9
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1	63.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31	...	67.0	7.13	65.0	33.6	454.3
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998	...	7.0	6.52	68.0	36.7	453.3
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304	...	73.0	6.53	71.0	39.8	57.3
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529	...	76.0	6.16	75.0	42.1	548.5
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483	...	78.0	7.10	78.0	43.5	547.3

2938 rows × 22 columns

Statistics:

#Loading Statistics

df.describe()

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	exp
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.000000	2938.000000	2919.000000	27
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.321247	42.035739	82.550188	
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.044034	160.445548	23.428046	
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000000	3.000000	
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.300000	0.000000	78.000000	
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	43.500000	4.000000	93.000000	
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	56.200000	28.000000	97.000000	
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000000	99.000000	

Shape of Data:

#Shape

df.shape

```
(2938, 22)
```

Data Info:

#Loading Info

df.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
```

Data Cleaning:

#Changing name of columns

df.rename(columns = {'Life expectancy ' : 'Life_expectancy' }, inplace = True)
df.rename(columns = {'Adult Mortality' : 'Adult_Mortality' }, inplace = True)
df.rename(columns = {'infant deaths' : 'infant_deaths' }, inplace = True)
df.rename(columns = {'percentage expenditure' : 'percentage_expenditure' }, inplace = True)
df.rename(columns = {'Hepatitis B' : 'Hepatiits_B' }, inplace = True)
df.rename(columns = {'Measles ' : 'Measles' }, inplace = True)
df.rename(columns = {'BMI ' : 'BMI' }, inplace = True)
df.rename(columns = {'under-five deaths ' : 'under_five_deaths' }, inplace = True)
df.rename(columns = {'Diphtheria ' : 'Diphtheria' }, inplace = True)
df.rename(columns = {'HIV/AIDS' : 'HIV_AIDS' }, inplace = True)
df.rename(columns = {'thinness 1-19 years' : 'thinness_1_to_19_years' }, inplace = True)
df.rename(columns = {'thinness 5-9 years' : 'thinness_5_to_9_years' }, inplace = True)
df.rename(columns = {'Income composition of resources' : 'Income_composition_of_resources' }, inplace= True)
df.rename(columns = {'Total expenditure' : 'Total_expenditure' }, inplace = True)

#Columns names after replacing

df.columns

```
Index(['Country', 'Year', 'Status', 'Life_expectancy', 'Adult_Mortality',
       'infant_deaths', 'Alcohol', 'percentage_expenditure', 'Hepatiits_B',
       'Measles', 'BMI', 'under_five_deaths', 'Polio', 'Total_expenditure',
       'Diphtheria', 'HIV_AIDS', 'GDP', 'Population', 'thinness_1_to_19_years',
       'thinness_5_to_9_years', 'Income_composition_of_resources',
       'Schooling'],
      dtype='object')
```

#Finding null values

for col in df.columns:
print(col , df[col].isnull().sum())

```
Country 0
Year 0
Status 0
Life_expectancy 10
Adult_Mortality 10
infant_deaths 0
Alcohol 194
percentage_expenditure 0
Hepatiits_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_to_19_years 34
thinness_5_to_9_years 34
Income_composition_of_resources 167
Schooling 163
```

#Droping null values

df.dropna(axis=0, inplace=True)
df

	Country	Year	Status	Life_expectancy	Adult_Mortality	infant_deaths	Alcohol	percentage_expenditure	Hepatiits_B	Measles	...	Polio	Total_exp
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31	...	67.0	
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998	...	7.0	
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304	...	73.0	
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529	...	76.0	
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483	...	78.0	

1649 rows × 22 columns

#Null values after cleaning

for col in df.columns:

print(col , df[col].isnull().sum())

```
Country 0
Year 0
Status 0
Life_expectancy 0
Adult_Mortality 0
infant_deaths 0
Alcohol 0
percentage_expenditure 0
Hepatiits_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_to_19_years 0
thinness_5_to_9_years 0
Income_composition_of_resources 0
Schooling 0
```

#Determine if data has any duplicate values

df.duplicated(subset=None,keep='first').sum()

```
0
```

No duplicate values exists

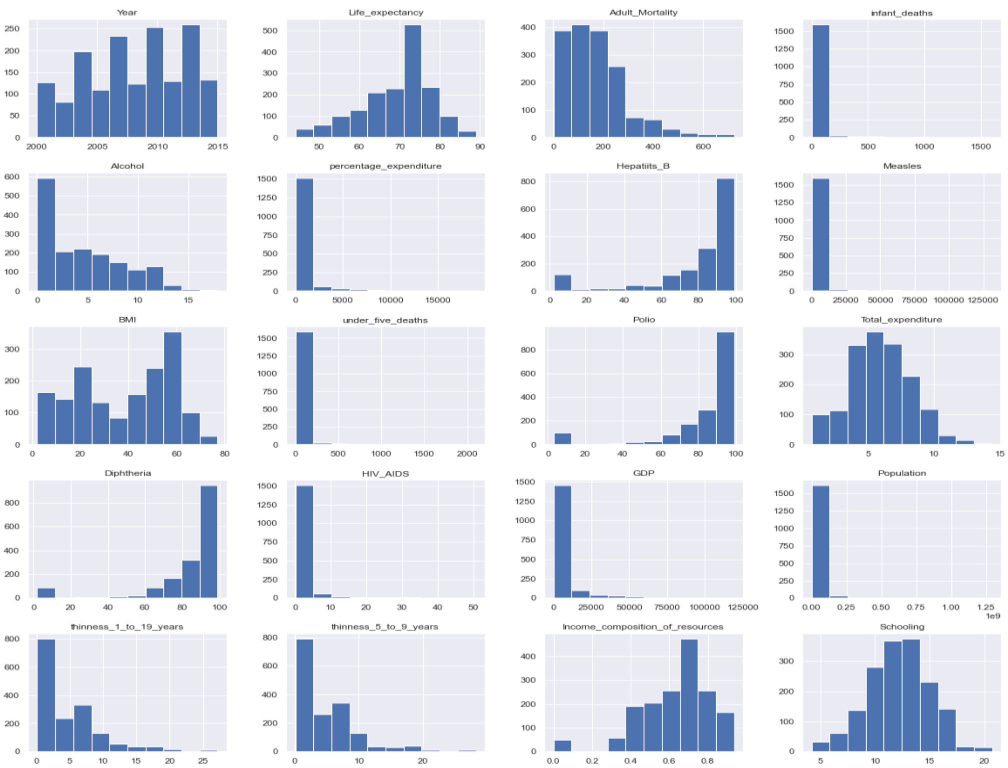
Data is clean.

EDA With Visualization:

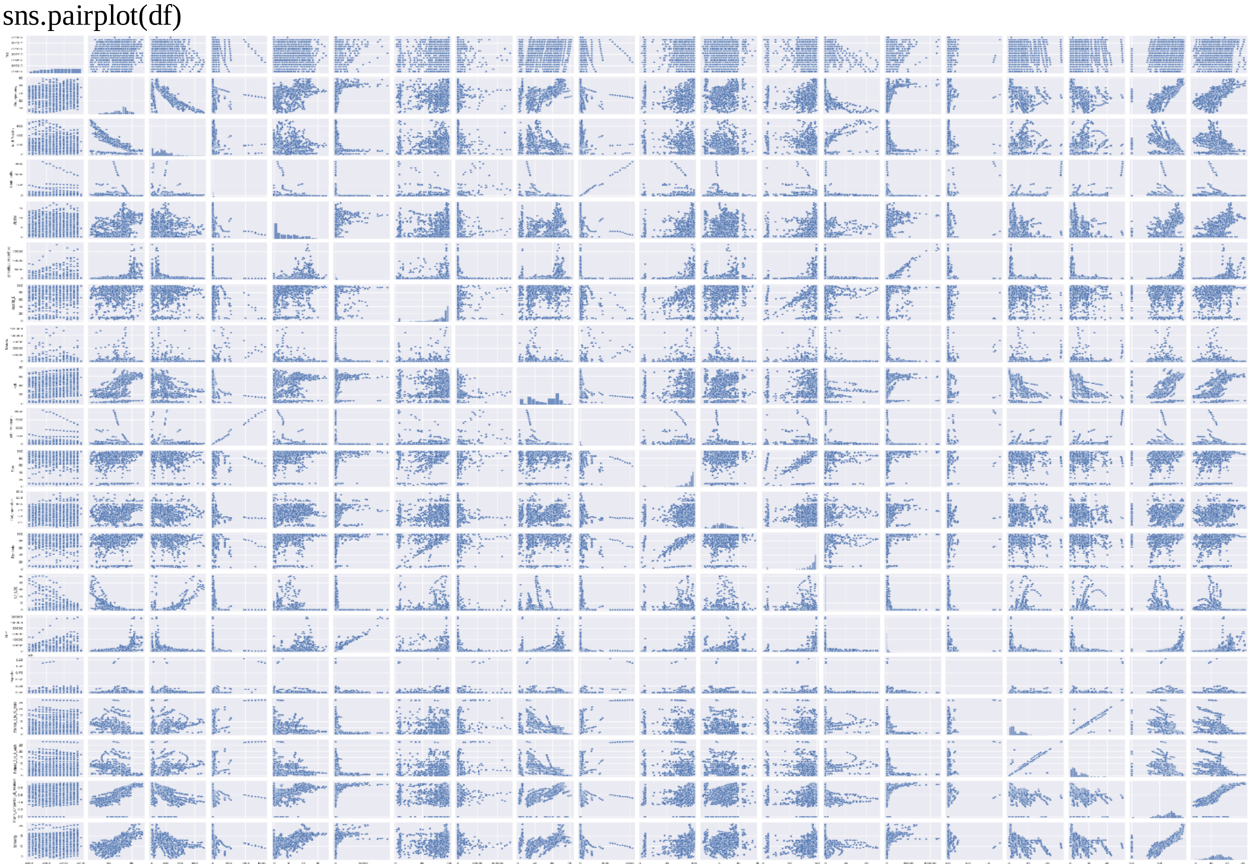
#Histograms of all columns

df.hist(figsize = (20, 20))

plt.show()



#Pairplot
sns.pairplot(df)



#Number of developing and developed cases
df.Status.value_counts().to_frame()

	Status
Developing	1407
Developed	242

#Visualizing number of developing and developed cases
from matplotlib import rcParams
rcParams['figure.figsize'] = 5,5
sns.countplot(x="Status", data= df,orient="v", color="pink")

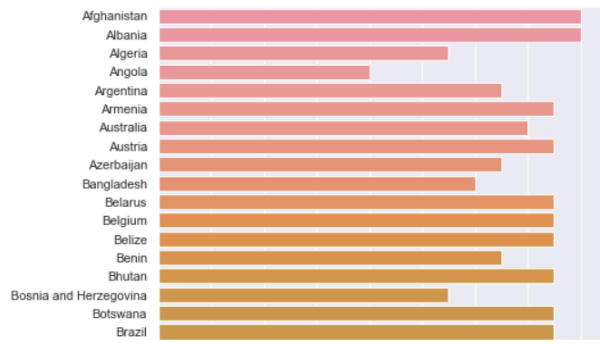
As we can see above many cases are in developing process

#Country Counts
df.Country.value_counts()

```
Afghanistan           16
Albania               16
Kiribati              15
Mexico                15
Mauritius             15
                      ..
Ireland                5
Sweden                 4
Netherlands            4
Haiti                  2
Equatorial Guinea      1
Name: Country, Length: 133, dtype: int64
```

#Visualize Country Counts
rcParams['figure.figsize'] = 7,40
sns.countplot(y="Country",data=df,orient="v")

```
Out[120]: <AxesSubplot:xlabel='count', ylabel='Country'>
```



#Health financing per capita health expenditure in each country

expenditure = df.groupby(by = 'Country')['Total_expenditure'].sum().sort_values(ascending = False).head(10).reset_index()

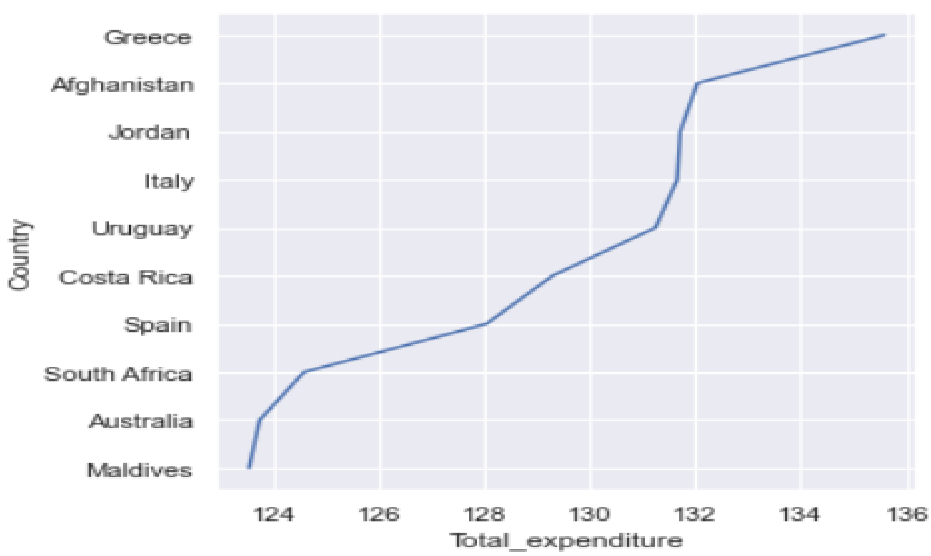
expenditure

	Country	Total_expenditure
0	Greece	135.58
1	Afghanistan	132.04
2	Jordan	131.72
3	Italy	131.66
4	Uruguay	131.25
5	Costa Rica	129.30
6	Spain	128.05
7	South Africa	124.59
8	Australia	123.75
9	Maldives	123.55

#Visualizing health financing per capita health expenditure in each country

rcParams['figure.figsize'] = 5,5

sns.lineplot(data=expenditure, x="Total_expenditure", y="Country")



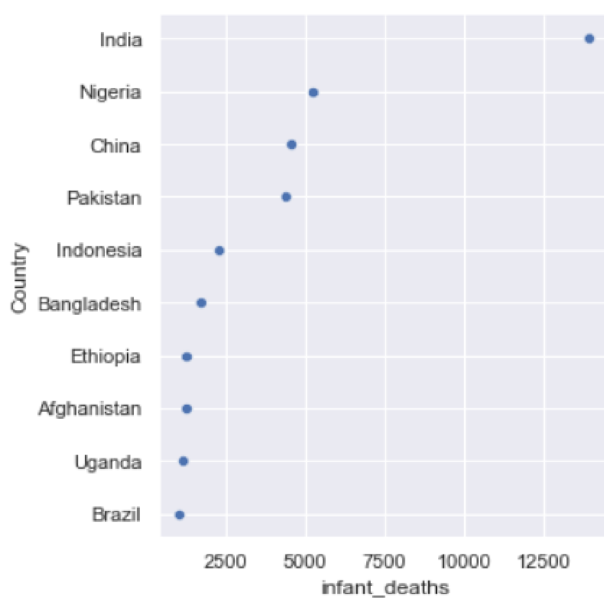
#Number of Infant Deaths per 1000 population

infant_deaths = df.groupby(by = 'Country')['infant_deaths'].sum().sort_values(ascending = False).head(10).reset_index()
infant_deaths

	Country	infant_deaths
0	India	13957
1	Nigeria	5237
2	China	4561
3	Pakistan	4402
4	Indonesia	2305
5	Bangladesh	1709
6	Ethiopia	1285
7	Afghanistan	1252
8	Uganda	1138
9	Brazil	1050

#Visualizing Number of Infant Deaths per 1000 population

sns.set_theme()
rcParams['figure.figsize'] = 10,5
sns.relplot(
data=infant_deaths,
x="infant_deaths", y="Country")

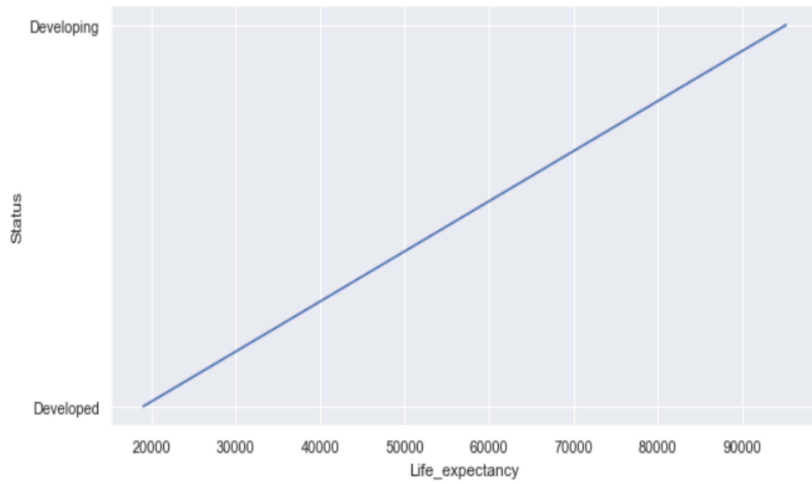


#Developed and developing cases life expectancy

groupby_country = df.groupby('Status')['Life_expectancy'].sum().sort_values(ascending = False).head(10).reset_index()
gc = pd.DataFrame(groupby_country)
gc

	Status	Life_expectancy
0	Developing	95236.1
1	Developed	19043.4

#Visualizing developed and developing cases life expectancy
sns.lineplot(data=gc, x="Life_expectancy", y="Status")



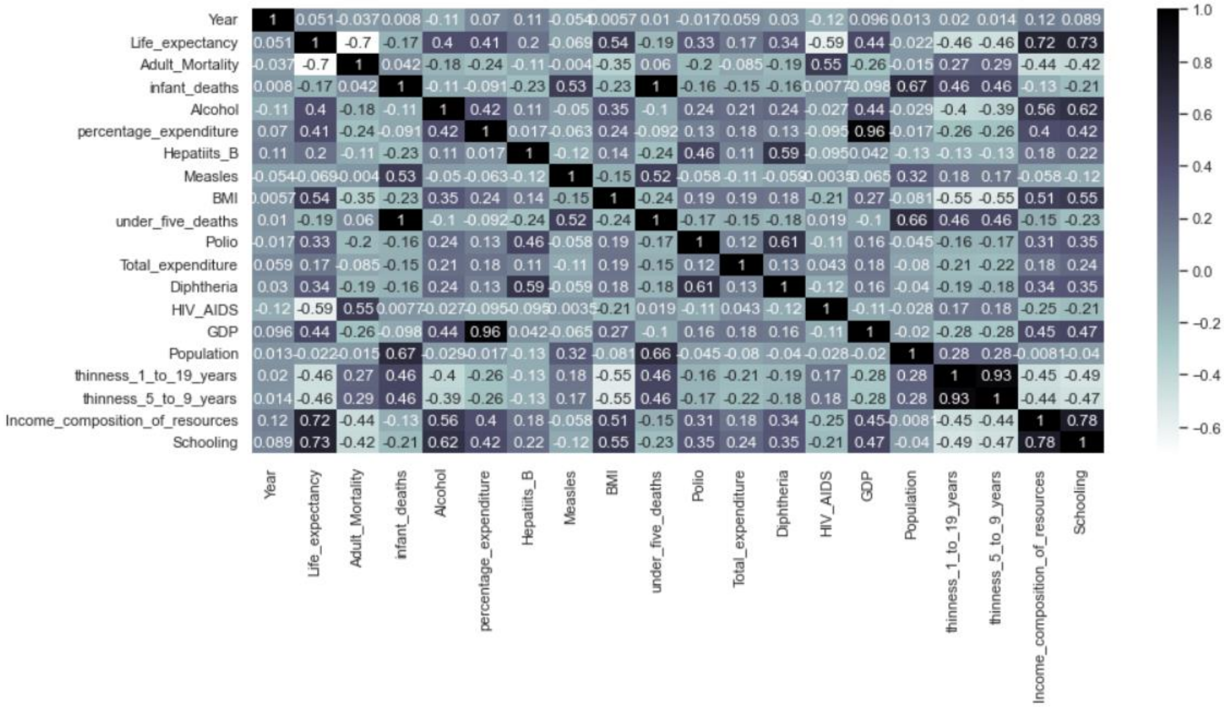
Correlation:

#Corelation
Corelation = df.corr()
Corelation

	Year	Life_expectancy	Adult_Mortality	infant_deaths	Alcohol	percentage_expenditure	Hepatiits_B	Measles	BM
Year	1.000000	0.050771	-0.037092	0.008029	-0.113365	0.069553	0.114897	-0.053822	0.00573
Life_expectancy	0.050771	1.000000	-0.702523	-0.169074	0.402718	0.409631	0.199935	-0.068881	0.54204
Adult_Mortality	-0.037092	-0.702523	1.000000	0.042450	-0.175535	-0.237610	-0.105225	-0.003967	-0.35154
infant_deaths	0.008029	-0.169074	0.042450	1.000000	-0.106217	-0.090765	-0.231769	0.532680	-0.23442
Alcohol	-0.113365	0.402718	-0.175535	-0.106217	1.000000	0.417047	0.109889	-0.050110	0.35339
percentage_expenditure	0.069553	0.409631	-0.237610	-0.090765	0.417047	1.000000	0.016760	-0.063071	0.24273
Hepatiits_B	0.114897	0.199935	-0.105225	-0.231769	0.109889	0.016760	1.000000	-0.124800	0.14330
Measles	-0.053822	-0.068881	-0.003967	0.532680	-0.050110	-0.063071	-0.124800	1.000000	-0.15324
BMI	0.005739	0.542042	-0.351542	-0.234425	0.353396	0.242738	0.143302	-0.153245	1.00000
under_five_deaths	0.010479	-0.192265	0.060365	0.996906	-0.101082	-0.092158	-0.240766	0.517506	-0.24213
Polio	-0.016699	0.327294	-0.199853	-0.156929	0.240315	0.128626	0.463331	-0.057850	0.18626
Total_expenditure	0.059493	0.174718	-0.085227	-0.146951	0.214885	0.183872	0.113327	-0.113583	0.18946
Diphtheria	0.029641	0.341331	-0.191429	-0.161871	0.242951	0.134813	0.588990	-0.058606	0.17629
HIV_AIDS	-0.123405	-0.592236	0.550691	0.007712	-0.027113	-0.095085	-0.094802	-0.003522	-0.21089
GDP	0.096421	0.441322	-0.255035	-0.098092	0.443433	0.959299	0.041850	-0.064768	0.26611
Population	0.012567	-0.022305	-0.015012	0.671758	-0.028880	-0.016792	-0.129723	0.321946	-0.08141
thinness_1_to_19_years	0.019757	-0.457838	0.272230	0.463415	-0.403755	-0.255035	-0.129406	0.180642	-0.54701
thinness_5_to_9_years	0.014122	-0.457508	0.286723	0.461908	-0.386208	-0.255635	-0.133251	0.174946	-0.55409
Income_composition_of_resources	0.122892	0.721083	-0.442203	-0.134754	0.561074	0.402170	0.184921	-0.058277	0.51050
Schooling	0.088732	0.727630	-0.421171	-0.214372	0.616975	0.422088	0.215182	-0.115660	0.55484

#Visualizing Corelation By Heatmap
top_fatures = Corelation.index
plt.figure(figsize = (14 , 6))

g = sns.heatmap(df[top_fatures].corr() , annot = True , cmap ="bone_r")



Separating Dependent and Independent Variables:

from sklearn.model_selection import train_test_split
x_data = df[['Country', 'Year', 'Status', 'Adult_Mortality', 'infant_deaths', 'Alcohol', 'percentage_expenditure', 'Hepatiits_B', 'Measles', 'BMI', 'under_five_deaths', 'Polio', 'Total_expenditure', 'Diphtheria', 'HIV_AIDS', 'GDP', 'Population', 'thinness_1_to_19_years', 'thinness_5_to_9_years', 'Income_composition_of_resources', 'Schooling']]
y_data = df[['Life_expectancy']]

Label Encoder:

data = x_data
categ = list(data.select_dtypes(include=['object']).columns)

#Encode Categorical Columns
le = preprocessing.LabelEncoder()
data[categ] = data[categ].apply(le.fit_transform)

#Changing datatypes of object
bools = list(data.select_dtypes(include=['bool']).columns)
data[bools] = data[bools].astype(int)

#Data info after datatype conversion
data.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1649 entries, 0 to 2937
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Country                         1649 non-null   int32  
 1   Year                            1649 non-null   int64  
 2   Status                          1649 non-null   int32  
 3   Adult_Mortality                 1649 non-null   float64
 4   infant_deaths                   1649 non-null   int64  
 5   Alcohol                         1649 non-null   float64
 6   percentage_expenditure          1649 non-null   float64
 7   Hepatiits_B                     1649 non-null   float64
 8   Measles                         1649 non-null   int64  
 9   BMI                             1649 non-null   float64
 10  under_five_deaths               1649 non-null   int64  
 11  Polio                           1649 non-null   float64
 12  Total_expenditure               1649 non-null   float64
 13  Diphtheria                      1649 non-null   float64
 14  HIV_AIDS                        1649 non-null   float64
 15  GDP                             1649 non-null   float64
 16  Population                      1649 non-null   float64
 17  thinness_1_to_19_years          1649 non-null   float64
 18  thinness_5_to_9_years           1649 non-null   float64
 19  Income_composition_of_resources 1649 non-null   float64
 20  Schooling                       1649 non-null   float64
dtypes: float64(15), int32(2), int64(4)
memory usage: 335.1 KB
```

#Data after Encoding

	Country	Year	Status	Adult_Mortality	infant_deaths	Alcohol	percentage_expenditure	Hepatiits_B	Measles	BMI	...	Polio	Total_expenditure	Diphtheri
0	0	2015	1	263.0	62	0.01	71.279624	65.0	1154	19.1	...	6.0	8.16	65.
1	0	2014	1	271.0	64	0.01	73.523582	62.0	492	18.6	...	58.0	8.18	62.
2	0	2013	1	268.0	66	0.01	73.219243	64.0	430	18.1	...	62.0	8.13	64.
3	0	2012	1	272.0	69	0.01	78.184215	67.0	2787	17.6	...	67.0	8.52	67.
4	0	2011	1	275.0	71	0.01	7.097109	68.0	3013	17.2	...	68.0	7.87	68.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2933	132	2004	1	723.0	27	4.36	0.000000	68.0	31	27.1	...	67.0	7.13	65.
2934	132	2003	1	715.0	26	4.06	0.000000	7.0	998	26.7	...	7.0	6.52	68.
2935	132	2002	1	73.0	25	4.43	0.000000	73.0	304	26.3	...	73.0	6.53	71.
2936	132	2001	1	686.0	25	1.72	0.000000	76.0	529	25.9	...	76.0	6.16	75.
2937	132	2000	1	665.0	24	1.68	0.000000	79.0	1483	25.5	...	78.0	7.10	78.

1649 rows × 21 columns

Min Max Scaler:

#Define min max scaler

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

#Transform data

scaled_x = scaler.fit_transform(data)

print(scaled_x)

#Scaled Data

	Country	Year	Status	Adult_Mortality	infant_deaths	Alcohol	percentage_expenditure	Hepatiits_B	Measles	BMI	...	Polio	Total_expenditure	Diphtheri
0	0	2015	1	263.0	62	0.01	71.279624	65.0	1154	19.1	...	6.0	8.16	65.
1	0	2014	1	271.0	64	0.01	73.523582	62.0	492	18.6	...	58.0	8.18	62.
2	0	2013	1	268.0	66	0.01	73.219243	64.0	430	18.1	...	62.0	8.13	64.
3	0	2012	1	272.0	69	0.01	78.184215	67.0	2787	17.6	...	67.0	8.52	67.
4	0	2011	1	275.0	71	0.01	7.097109	68.0	3013	17.2	...	68.0	7.87	68.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2933	132	2004	1	723.0	27	4.36	0.000000	68.0	31	27.1	...	67.0	7.13	65.
2934	132	2003	1	715.0	26	4.06	0.000000	7.0	998	26.7	...	7.0	6.52	68.
2935	132	2002	1	73.0	25	4.43	0.000000	73.0	304	26.3	...	73.0	6.53	71.
2936	132	2001	1	686.0	25	1.72	0.000000	76.0	529	25.9	...	76.0	6.16	75.
2937	132	2000	1	665.0	24	1.68	0.000000	79.0	1483	25.5	...	78.0	7.10	78.

1649 rows × 21 columns

Train Test Split:

x_train , x_test , y_train , y_test = train_test_split (scaled_x , y_data , test_size = 0.3 , train_size = 0.7 , random_state = 0)

#Shape Of Train Test Data
print ("Train Data x Shape: " , x_train.shape)
print ("Test Data x Shape: " , x_test.shape)
print ("Train Data y Shape: " , y_train.shape)
print ("Test Data y Shape: " , y_test.shape)

```
[[0.         1.         1.         ... 0.61209964 0.51175214 0.35757576]
 [0.         0.93333333 1.         ... 0.61921708 0.50854701 0.35151515]
 [0.         0.86666667 1.         ... 0.62633452 0.50213675 0.34545455]
 ...
 [1.         0.13333333 1.         ... 0.04270463 0.45619658 0.35151515]
 [1.         0.06666667 1.         ... 0.0569395  0.45619658 0.33939394]
 [1.         0.         1.         ... 0.39501779 0.46367521 0.33939394]]
```

Linear regression:

from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)

```
LinearRegression()
```

#Predicted values by model
pre_lr = reg.predict(x_test)
pre_lr

```
Out[155]: array([[59.94207222],
       [75.59419528],
       [71.20771619],
       [55.49292841],
       [61.90232091],
       [83.84991803],
       [49.94769265],
       [61.50636378],
       [42.5009569 ],
       [74.1670219 ],
       [76.33024977],
       [74.26737913],
       [78.5668246 ],
       [70.5044872 ],
       [84.1388897 ],
       [45.10475129],
       [46.24288454],
       [68.89741717],
       [73.73843575],
```

#Score
reg.score(x_test,y_test)*100

```
84.29368282860217
```

#MSE, MAE and R^2
print('R^2 : ',metrics.r2_score(y_test, pre_lr))
print('Mean Absolute Error : ',metrics.mean_absolute_error(y_test, pre_lr))

print('Mean Squared Error : ',metrics.mean_squared_error(y_test, pre_lr))

```
R^2 :  0.8429368282860217
Mean Absolute Error :  2.7673355518522182
Mean Squared Error :  12.862871599744555
```

Bagging Regressor:

#Importing bagging regressor
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor
breg = BaggingRegressor(base_estimator=SVR(), n_estimators=10, random_state=0)
breg.fit(x_train, y_train)

```
BaggingRegressor(base_estimator=SVR(), random_state=0)
```

#Predicted values by model
pre_bag = breg.predict(x_test)
pre_bag

```
Out[161]: array([60.48233193, 76.51586198, 72.35492508, 57.06573814, 60.77621667,
       81.8031891 , 51.31215286, 63.28381723, 54.12616072, 75.96457477,
       78.95213076, 74.70124239, 79.60843352, 71.30821013, 79.52594929,
       56.97849581, 57.33282053, 68.50233095, 73.66360434, 70.47901861,
       79.5654534 , 70.08665009, 64.42214119, 72.92434911, 73.02040779,
       65.67047093, 64.0238549 , 63.91789431, 78.9162672 , 69.08110317,
       71.10720823, 52.03594797, 57.56429635, 71.44119226, 72.2333887 ,
       71.64229778, 69.59210664, 79.01946473, 79.36862454, 56.22532062,
       78.62987124, 64.88568559, 78.18641113, 74.66773215, 82.19243941,
       80.21253472, 55.7576349 , 71.98943283, 70.47658   , 79.77017428,
       63.09419262, 69.45781308, 58.41270938, 77.72271795, 68.85269156,
       79.35972368, 74.1645692 , 75.63891626, 68.82521117, 67.81294713,
       72.83860345, 69.27226563, 68.91877727, 76.46512459, 76.2777936 ,
       78.62198655, 71.83668705, 69.98479624, 71.53842303, 73.24005741,
       67.52584315, 79.9612721 , 47.42169892, 78.73948453, 78.17972053,
       63.92733141, 77.52042954, 78.12609551, 64.86055145, 66.19194794,
       61.23627015, 75.66140539, 64.24550195, 67.37967658, 61.9923026 ,
       69.36620224, 72.44881551, 45.67871109, 69.34667979, 69.47605839,
       62.87993568, 74.44652705, 75.74339987, 51.46834518, 69.14711564,
```

#MAE, MSE and R^2
print('R^2 : ',metrics.r2_score(y_test, pre_bag))
print('Mean Absolute Error : ',metrics.mean_absolute_error(y_test, pre_bag))
print('Mean Squared Error : ',metrics.mean_squared_error(y_test, pre_bag))

```
R^2 :  0.8320881765876771
Mean Absolute Error :  2.670106329928039
Mean Squared Error :  13.751334581252893
```

XG Boost Predictions with Full data:

#Loading xgboost
import xgboost as xg
fullxg = xg.XGBRegressor(objective ='reg:linear', n_estimators = 10, seed = 123)

#Fitting the model
xgfull = fullxg.fit(x_train, y_train)

#Predicted values of model

predfull = fullxg.predict(scaled_x)
predfull

```
array([61.75452 , 57.646152, 59.42004 , ..., 43.518574, 44.540825,
       44.867332], dtype=float32)
```

#MSE, MAE and R^2
print('R^2 : ',metrics.r2_score(y_data, predfull))
print('Mean Absolute Error : ',metrics.mean_absolute_error(y_data, predfull))
print('Mean Squared Error : ',metrics.mean_squared_error(y_data, predfull))

```
R^2 :  0.8996935641510315
Mean Absolute Error :  2.2047304536300114
Mean Squared Error :  7.757435219573645
```

#Saving model using joblib
import joblib
joblib.dump(xgfull, 'model_save2')
model2 = joblib.load('model_save2')

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=10, n_jobs=8,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=123, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             seed=123, subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

#Input data for train model
d1 = ["Bahamas", 1999, "Developed", 43, 65, 76, 56, 78, 67, 2.4, 76, 765, 786, 76, 6.6, 76, 7645345678899888, 56, 89, 7896, 98]
d2 = np.array([d1])
data1 = pd.DataFrame(d2)
data1

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	Bahamas	1999	Developed	43	65	76	56	78	67	2.4	...	765	786	76	6.6	76	7645345678899888	56	89	7896	98

#Label Encoder and min max scaler to convert and normalize input data
categ = list(data1.select_dtypes(include=['object']).columns)
le = preprocessing.LabelEncoder()
data1[categ] = data1[categ].apply(le.fit_transform)
bools = list(data1.select_dtypes(include=['bool']).columns)
data1[bools] = data1[bools].astype(int)
scaler1 = MinMaxScaler()
scaled_data1 = scaler.fit_transform(data1)

scaled_data1

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

#Predicting Life Expectancy from train model using input data
model2.predict(scaled_data1)

```
array([50.865726], dtype=float32)
```

XG Boost Regression with Train and Test data:

#Predicted values of XG Boost
import xgboost as xg
xgb_r = xg.XGBRegressor(objective ='reg:linear', n_estimators = 10, seed = 123)
xg = xgb_r.fit(x_train, y_train)
pred = xgb_r.predict(x_test)
pred

```
[22:20:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:lin
ear is now deprecated in favor of reg:squarederror.

Out[144]: array([57.150448, 79.28581 , 73.38556 , 52.42246 , 59.91934 , 79.39506 ,
       52.22916 , 50.659386, 44.501724, 71.702705, 78.05668 , 71.81858 ,
       76.59998 , 69.76818 , 76.019424, 49.069935, 49.120934, 70.476204,
       70.84275 , 67.01421 , 76.002495, 64.511856, 63.30731 , 71.51044 ,
       71.45687 , 63.767136, 64.377975, 55.302467, 78.875916, 68.26204 ,
       69.34918 , 47.40433 , 59.35104 , 68.72463 , 71.91212 , 68.59195 ,
       72.601814, 80.58408 , 78.57061 , 56.412605, 73.47177 , 63.377087,
       79.010574, 66.75747 , 79.41991 , 79.1699  , 50.0022  , 73.38556 ,
       70.07894 , 79.82624 , 60.008205, 69.205   , 54.497154, 77.834724,
       65.330475, 80.54611 , 73.084785, 77.31941 , 65.99192 , 65.17439 ,
       70.84275 , 71.233696, 64.74178 , 73.58172 , 73.83459 , 79.13193 ,
       71.17783 , 70.58811 , 71.296646, 72.87761 , 65.66038 , 79.010574,
       45.08122 , 80.15629 , 78.491516, 53.32604 , 73.15613 , 80.19748 ,
       64.31973 , 66.141174, 59.581646, 73.48605 , 65.97903 , 65.24479 ,
       62.68722 , 64.62638 , 72.08066 , 43.812805, 63.664097, 65.63582 ,
       60.675854, 73.084785, 73.48605 , 45.25416 , 70.21816 , 59.369514,
       67.41278 , 56.129505, 46.30752 , 51.423492, 45.351658, 61.935707,
```

#Score
xgb_r.score(x_test,y_test)*100

```
89.10268654445191
```

#MAE, MSE and R^2
print('R^2 : ',metrics.r2_score(y_test, pred))
print('Mean Absolute Error : ',metrics.mean_absolute_error(y_test, pred))
print('Mean Squared Error : ',metrics.mean_squared_error(y_test, pred))

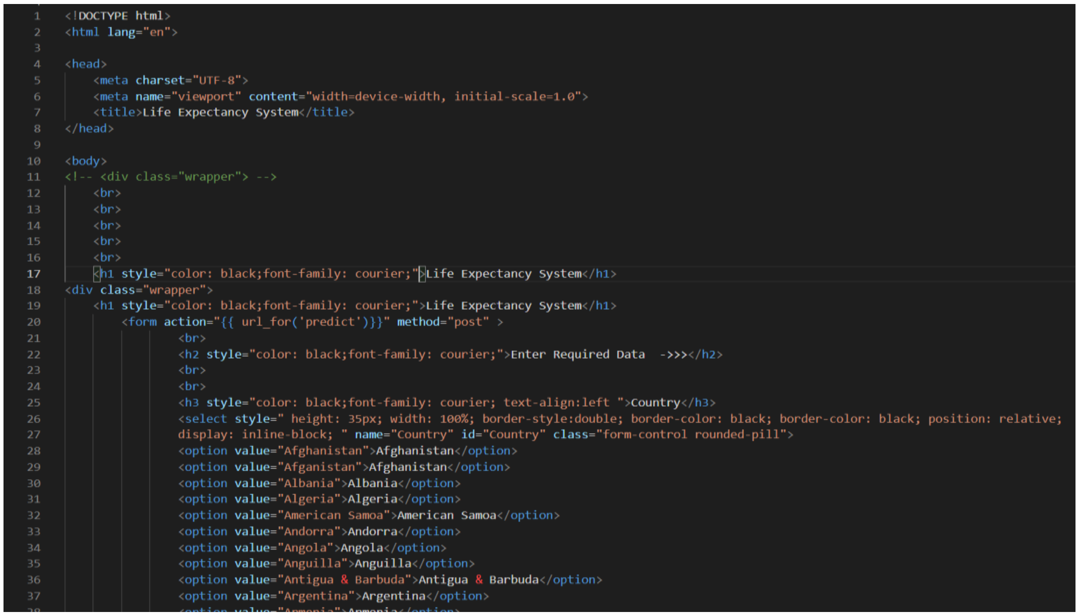
```
R^2 :  0.8910268654445191
Mean Absolute Error :  2.364579526342527
Mean Squared Error :  8.924481928592604
```

#Saving Model using Pickle
import pickle
file = open('xg.pkl', 'wb')
pickle.dump(xg, file)
#Deployment Using Flask Framework

```
from typing import Text
from flask import Flask, render_template, request
import requests
import pickle
import numpy as np
import pandas as pd
import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
app = Flask(__name__)
model = pickle.load(open('xg.pkl', 'rb'))
@app.route('/', methods=['GET'])
def Home():
 return render_template('home.html')
@app.route("/predict", methods=['POST'])
def predict():
Country = request.form.get('Country')
Year = request.form.get('Year')
Status = request.form.get('Status')
Adult_Mortality = request.form.get('Adult_Mortality')
infant_deaths = request.form.get('infant_deaths')
Alcohol = request.form.get('Alcohol')
percentage_expenditure = request.form.get('percentage_expenditure')
Hepatiits_B = request.form.get('Hepatiits_B')
Measles = request.form.get('Measles')
BMI = request.form.get('BMI')
under_five_deaths = request.form.get('under_five_deaths')
Polio = request.form.get('Polio')
Total_expenditure = request.form.get('Total_expenditure')
Diphtheria = request.form.get('Diphtheria')
HIV_AIDS = request.form.get('HIV_AIDS')
GDP = request.form.get('GDP')
Population = request.form.get('Population')
thinness_1_to_19_years = request.form.get('thinness_1_to_19_years')
thinness_5_to_9_years = request.form.get('thinness_5_to_9_years')
Income_composition_of_resources = request.form.get('Income_composition_of_resources')
Schooling = request.form.get('Schooling')
df = pd.DataFrame([[Country, Year, Status, Adult_Mortality, infant_deaths, Alcohol,
percentage_expenditure, Hepatiits_B, Measles, BMI, under_five_deaths, Polio, Total_expenditure,
Diphtheria, HIV_AIDS, GDP, Population, thinness_1_to_19_years, thinness_5_to_9_years,
Income_composition_of_resources, Schooling]])
data = df
categ = list(data.select_dtypes(include=['object']).columns)
le = preprocessing.LabelEncoder()
data[categ] = data[categ].apply(le.fit_transform)
bools = list(data.select_dtypes(include=['bool']).columns)
data[bools] = data[bools].astype(int)
```

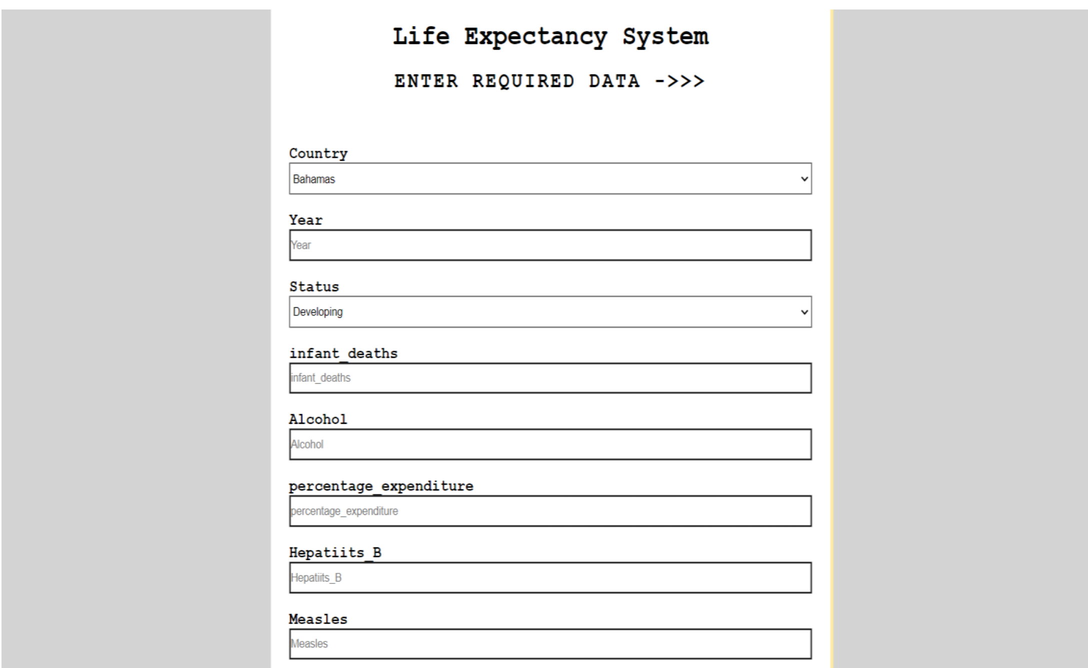
```
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
output = model.predict(df)
return render_template('home.html', prediction_text='Life Expectancy {}'.format(output)
if __name__ == "__main__":
    app.run(debug=True)
```

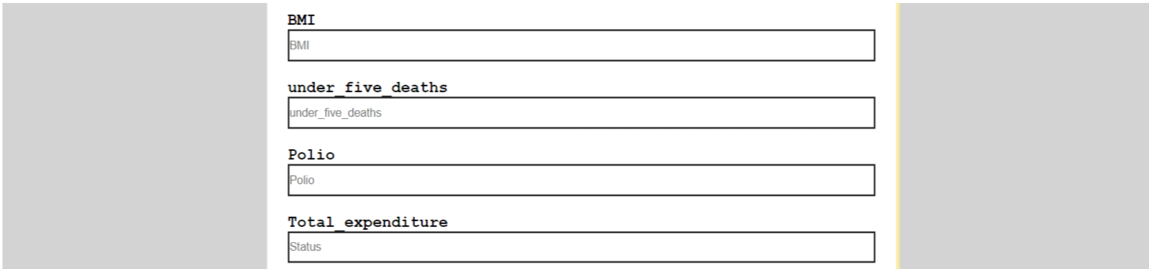
#HTML View

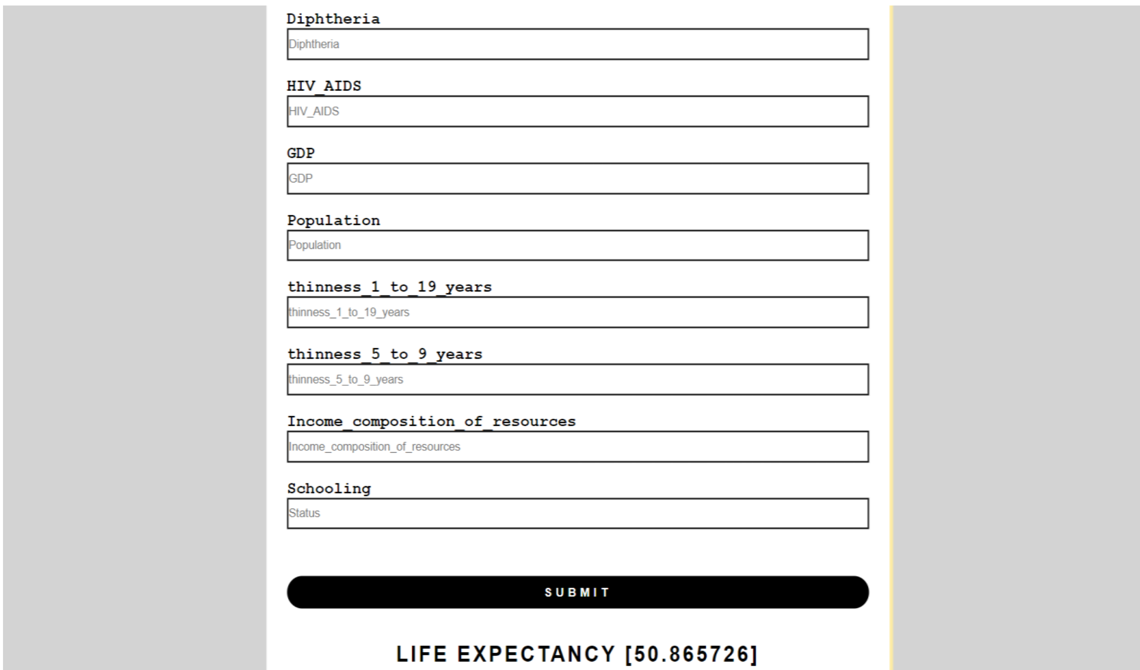


```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Life Expectancy System</title>
</head>

<body>
<!-- <div class="wrapper"> -->
    <br>
    <br>
    <br>
    <br>
    <br>
    <h1 style="color: black;font-family: courier;">Life Expectancy System</h1>
<div class="wrapper">
    <h1 style="color: black;font-family: courier;">Life Expectancy System</h1>
        <form action="{{ url_for('predict')}}" method="post" >
                <br>
                <h2 style="color: black;font-family: courier;">Enter Required Data  ->>></h2>
                <br>
                <br>
                <h3 style="color: black;font-family: courier; text-align:left ">Country</h3>
                <select style=" height: 35px; width: 100%; border-style:double; border-color: black; border-color: black; position: relative;
                display: inline-block; " name="Country" id="Country" class="form-control rounded-pill">
                <option value="Afghanistan">Afghanistan</option>
                <option value="Afganistan">Afghanistan</option>
                <option value="Albania">Albania</option>
                <option value="Algeria">Algeria</option>
                <option value="American Samoa">American Samoa</option>
                <option value="Andorra">Andorra</option>
                <option value="Angola">Angola</option>
                <option value="Anguilla">Anguilla</option>
                <option value="Antigua & Barbuda">Antigua & Barbuda</option>
                <option value="Argentina">Argentina</option>
```

#Web Interface







Conclusion:

I make a web-based prediction system which will predict life expectancy of every country. This system is useful for Government and Healthcare sectors for the betterment of society. Government sector can predict their life expectancy for there country and make good decision. Like they should increase charity center because many people are dying because of poverty, or they should enlarge health care systems and their resources for people of society. They can try making organizations for poor people and feed them on daily bases and educate them for countries better future. They perform action against alcoholics in terms of stopping import and making of alcohol in their countries because alcohol consumption is the big cause of decrease in life expectancy. They increase health expenditure for more reliability of there people. They should perform actions against polio which is main cause of low life expectancy of a country. They should arrange more resources for diseases like HIV aids, Hepatitis b and Diphtheria in their country.