Importing Data & Required Libraries

In [1]:
```
# Import important Libraries
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
import numpy as np
import plotly.express as px
from sklearn import preprocessing
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import utils
import matplotlib.pyplot as pyplot
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.model_selection import KFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
import warnings
from sklearn.impute import KNNImputer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import math
from sklearn.linear_model import Ridge
from sklearn.kernel_ridge import KernelRidge
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn import decomposition, datasets
from sklearn.metrics import fbeta_score, make_scorer
warnings.filterwarnings('ignore')

# Display bigger size of data frames
pd.set_option('display.max_columns', 500)
pd.set_option('display.min_row', 5505)
```

In [2]:
```
df = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')


df = pd.concat([df,test])
# df = df.drop(['ID'], axis = 1)
```

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

Out[3]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Popu
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1	83	6.0	8.16	65.0	0.1	584.259210	3373
1	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1	89	62.0	8.13	64.0	0.1	631.744976	3173
2	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6	93	67.0	8.52	67.0	0.1	669.959000	369
3	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2	97	68.0	7.87	68.0	0.1	63.537231	297
4	Afghanistan	2010	Developing	58.8	279.0	74	0.01	79.679367	66.0	1989	16.7	102	66.0	9.20	66.0	0.1	553.328940	288

In [4]:
```
# Shape of the data
print('Shape of the Data Set: ', df.shape)
print('Data Columns: ', df.columns)
```

```
Shape of the Data Set:  (2937, 23)
Data Columns:  Index(['Country', 'Year', 'Status', 'Life expectancy ', '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', 'ID'],
      dtype='object')
```

Descriptive Statistics

- Descriptive Statistics is one of the most Important step to understand the data and take out insights.
- First we will describe the statistics for the Numerical columns.
- For numerical columns we check for stats such as Max, Min, Mean, Count, Standart Deviation, 25 percentile, 50 percentile, 75 percentile.
- Then, we will check for the Descriptive statistics for Categorical columns.
- For categorical columns we check for stats such a Count, Frequency, Top & Unique Elements.
- Mostly, we use encoding (any kind of encoding) for categorical data.

In [5]:
```
# Before getting deep into the problem, let's try to get some descriptive statistics for numerical columns
df.describe().style.background_gradient(cmap = 'copper')
```

Out[5]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphth
count	2937.000000	2333.000000	2928.000000	2937.000000	2743.000000	2937.000000	2384.000000	2937.000000	2903.000000	2937.000000	2918.000000	2711.000000	2918.000
mean	2007.516854	69.302443	164.796448	30.314266	4.604535	738.502658	80.933305	2420.416071	38.305925	42.050051	82.544894	5.939059	82.318
std	4.613517	9.514162	124.292079	117.945256	4.052202	1988.206672	25.072839	11469.138238	20.030471	160.470994	23.430315	2.498370	23.719
min	2000.000000	39.000000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000000	3.000000	0.370000	2.000
25%	2004.000000	63.500000	74.000000	0.000000	0.880000	4.720594	77.000000	0.000000	19.300000	0.000000	78.000000	4.260000	78.000
50%	2008.000000	72.100000	144.000000	3.000000	3.760000	64.969645	92.000000	17.000000	43.500000	4.000000	93.000000	5.760000	93.000
75%	2012.000000	75.800000	228.000000	22.000000	7.705000	441.844624	97.000000	361.000000	56.200000	28.000000	97.000000	7.495000	97.000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000000	99.000000	17.600000	99.000

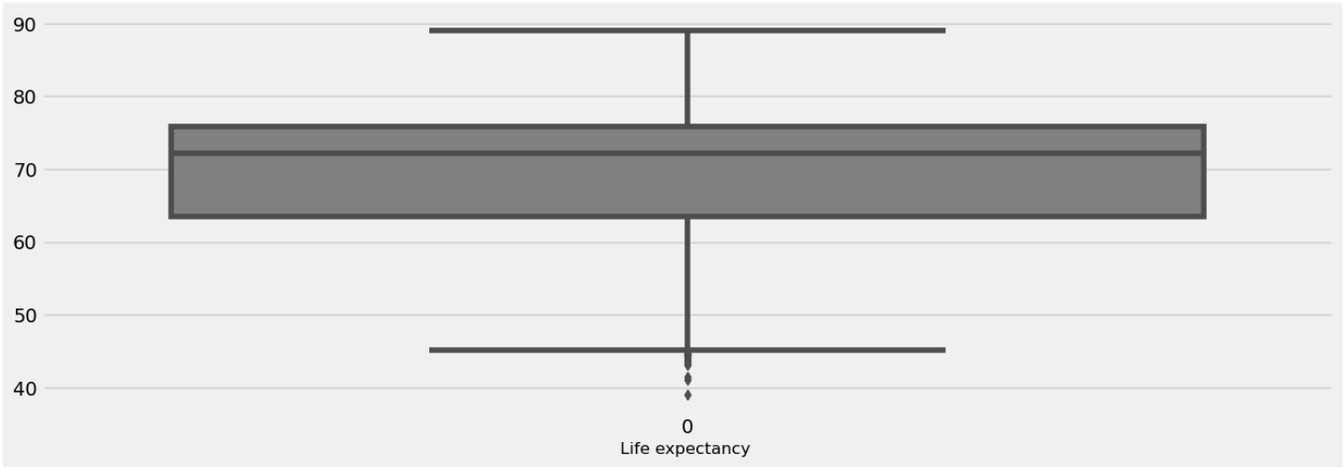
In [6]:
```
# Now, let's get the picture of all the categorical columns in our data
df.describe(include = 'object').style.background_gradient(cmap = 'copper')
```

Out[6]:

	Country	Status
count	2937	2937
unique	192	2
top	Afghanistan	Developing
freq	16	2425

In [7]:
```
# Because we want to predict the Life Expectancy, let's check the Target Class Balance

# check the boxplots for the target class
plt.rcParams['figure.figsize'] = (15, 5)
plt.style.use('fivethirtyeight')

# Box plot for Life expectancy

sns.boxplot(df['Life expectancy '].values, color = 'grey')
plt.xlabel('Life expectancy ', fontsize = 12)
print(f"The life expectancy average is: {df['Life expectancy '].mean()}, and the standard deviation is: {df['Life expectancy '].std()}")
```

```
The life expectancy average is: 69.30244320617236, and the standard deviation is: 9.514161514729253
```



From the above chart, we can see the distribution of the life expectancy; Where the average is around 69 years and the standard deviation is 9.5.

Treatment of Missing Values

Now, we would like to check whether null values exist in our data.

- Treatment of Missing Values is very Important Step in any Machine Learning Model Creation
- Missing Values can be cause due to various reasons such as the filling incomplete forms, values not available, etc
- There are so many types of Missing Values such as
  - Missing values at Random
  - Missing values at not Random
  - Missing Values at Completely Random
- What can we do to Impute or Treat Missing values to make a Good Machine Learning Model
  - We can use Business Logic to Impute the Missing Values
  - We can use Statistical Methods such as Mean, Median, and Mode.
  - We can use ML Techniques to impute the Missing values
  - We can delete the Missing values, when the Missing values percentage is very High.
- When to use Mean, and when to use Median?
  - We use Mean, when we do not have Outliers in the dataset for the Numerical Variables.
  - We use Median, when we have outliers in the dataset for the Numerical Variables.
  - We use Mode, When we have Categorical Variables.

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Country                             0
Year                                0
Status                              0
Life expectancy                   604
Adult Mortality                     9
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                               447
Population                        651
 thinness  1-19 years              34
 thinness 5-9 years                34
Income composition of resources   166
Schooling                         162
ID                               2342
dtype: int64
```

In [9]:
```
# missing values in data set

# calculate the total missing values in the dataset
train_total = df.isnull().sum()

# calculate the percentage of missing values in the dataset
train_percent = ((df.isnull().sum()/df.shape[0])*100).round(2)

# make a dataset consisting of total no. of missing values and percentage of missing values in the dataset
train_missing_data = pd.concat([train_total, train_percent],
                               axis=1,
                               keys=['Total missing values', '% of missing values'],
                               sort = True)

train_missing_data.style.background_gradient(cmap = 'BuGn')
```

Out[9]:

	Total missing values	% of missing values
BMI	34	1.160000
HIV/AIDS	0	0.000000
thinness 1-19 years	34	1.160000
thinness 5-9 years	34	1.160000
Adult Mortality	9	0.310000
Alcohol	194	6.610000
Country	0	0.000000
Diphtheria	19	0.650000
GDP	447	15.220000
Hepatitis B	553	18.830000
ID	2342	79.740000
Income composition of resources	166	5.650000
Life expectancy	604	20.570000
Measles	0	0.000000
Polio	19	0.650000
Population	651	22.170000
Schooling	162	5.520000
Status	0	0.000000
Total expenditure	226	7.690000
Year	0	0.000000
infant deaths	0	0.000000
percentage expenditure	0	0.000000
under-five deaths	0	0.000000

Missing Values for Target Class - Life Expectancy

In [10]:
```
# df[df['Life expectancy '].isnull()]

df[df['Life expectancy '].isnull() & df['ID'].isnull()]
```

Out[10]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP
610	Dominica	2013	Developing	NaN	NaN	0	0.01	11.419555	96.0	0	58.4	0	96.0	5.58	96.0	0.1	722.756650
1313	Marshall Islands	2013	Developing	NaN	NaN	0	0.01	871.878317	8.0	0	81.6	0	79.0	17.24	79.0	0.1	3617.752354
1369	Monaco	2013	Developing	NaN	NaN	0	0.01	0.000000	99.0	0	NaN	0	99.0	4.30	99.0	0.1	NaN
1442	Nauru	2013	Developing	NaN	NaN	0	0.01	15.606596	87.0	0	87.3	0	87.0	4.65	87.0	0.1	136.183210
1523	Niue	2013	Developing	NaN	NaN	0	0.01	0.000000	99.0	0	77.3	0	99.0	7.20	99.0	0.1	NaN
1563	Palau	2013	Developing	NaN	NaN	0	NaN	344.690631	99.0	0	83.3	0	99.0	9.27	99.0	0.1	1932.122370
1739	Saint Kitts and Nevis	2013	Developing	NaN	NaN	0	8.54	0.000000	97.0	0	5.2	0	96.0	6.14	96.0	0.1	NaN

Upon observation we find that the country with missing life_expectancy, have data for only 1 year (2013).

So the most appropriate thing to do is impute it with mean of Developing Country, as these are developing countries located in Oceania.

However, these countries have a lot of other missing data. Hence, we decided to drop them

In [11]:
```
df.drop(df[df['Life expectancy '].isnull() & df['ID'].isnull()].index, inplace = True)
```

Missing Values for Alcohol & Schooling columns

In [12]:
```
df[df['Alcohol'].isnull()]
```

Out[12]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GD
40	Angola	2015	Developing	52.4	335.0	66	NaN	0.000000	64.0	118	23.3	98	7.0	NaN	64.0	1.9	3695.7937
67	Argentina	2015	Developing	76.3	116.0	8	NaN	0.000000	94.0	0	62.8	9	93.0	NaN	94.0	0.1	13467.1236
89	Australia	2015	Developed	82.8	59.0	1	NaN	0.000000	93.0	74	66.6	1	93.0	NaN	93.0	0.1	56554.3876
102	Austria	2015	Developed	81.5	65.0	0	NaN	0.000000	93.0	309	57.6	0	93.0	NaN	93.0	0.1	43665.9470
115	Azerbaijan	2015	Developing	72.7	118.0	5	NaN	0.000000	96.0	0	52.5	6	98.0	NaN	96.0	0.1	55.3138
124	Bahamas	2015	Developing	76.1	147.0	0	NaN	0.000000	95.0	0	64.5	0	95.0	NaN	95.0	0.1	Na
140	Bahrain	2015	Developing	76.9	69.0	0	NaN	0.000000	98.0	0	63.6	0	98.0	NaN	98.0	0.1	22688.8782
172	Belarus	2015	Developing	72.3	196.0	0	NaN	0.000000	99.0	2	62.3	0	99.0	NaN	99.0	0.1	5949.1167
184	Belgium	2015	Developed	81.1	74.0	0	NaN	0.000000	98.0	47	63.7	1	99.0	NaN	99.0	0.1	4356.8750

All the data for alcohol missing is from 2015

In [13]:
```
df[(df['Status'] == 'Developed') &
  (df['Alcohol'].notna())]['Alcohol'].mean()
```

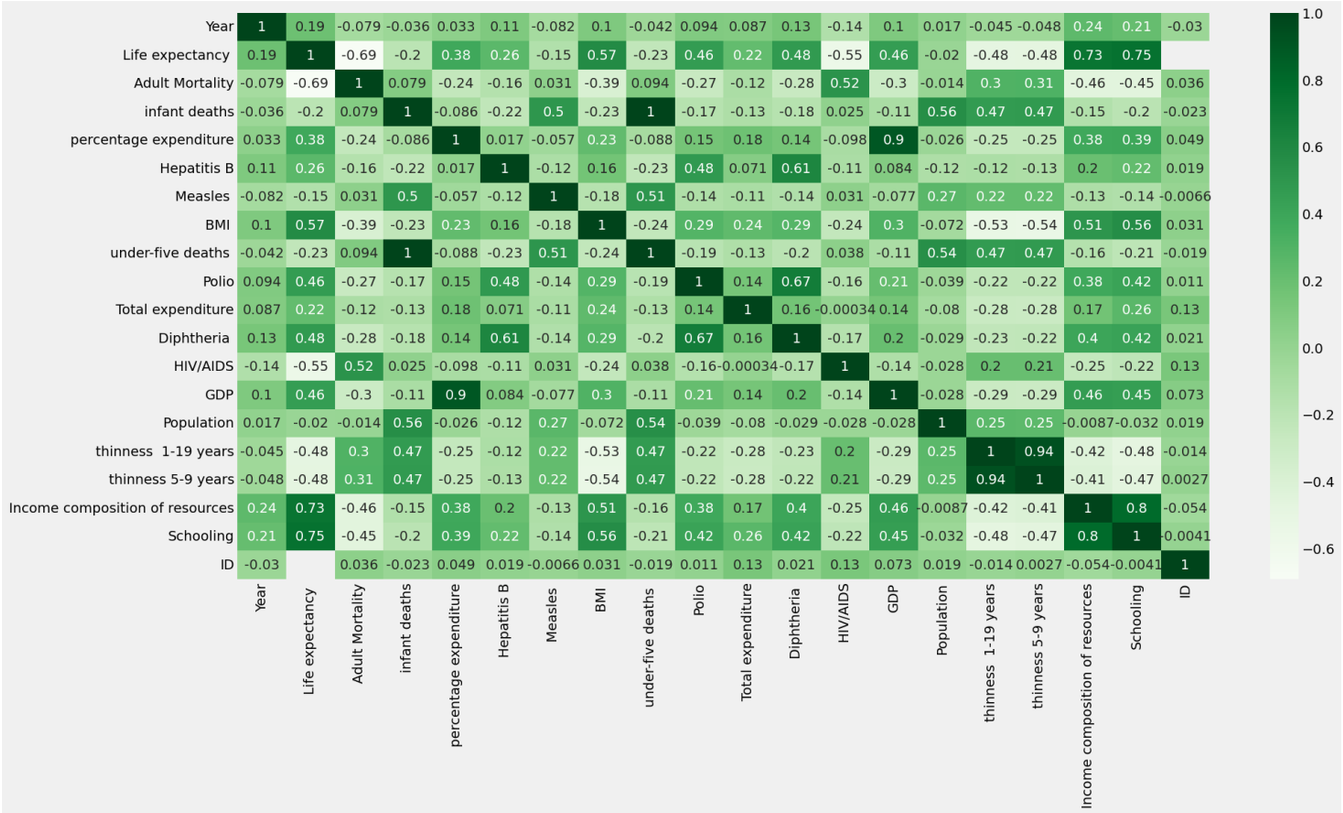
Out[13]:
```
9.826735537190078
```

In [14]:
```
df = df.reset_index()
df = df.drop(['index'], axis = 1)
```

In [15]:
```
df = df.drop(['Alcohol'], axis = 1)
```

In [16]:
```
# df['Alcohol'] = np.where(((df['Status'] == 'Developing') &
#         (df['Alcohol'].isna())), df[(df['Status'] == 'Developing') & (df['Alcohol'].notna())]['Alcohol'].mean(),
#         df['Alcohol'])

# df['Alcohol'] = np.where(((df['Status'] == 'Developed') &
#         (df['Alcohol'].isna())), df[(df['Status'] == 'Developed') & (df['Alcohol'].notna())]['Alcohol'].mean(),
#         df['Alcohol'])
```

In [17]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True,cmap='Greens')
plt.show()
```



In [18]:
```
# # lets understand the impact of Alcohol on Schooling
# px.scatter(df, y = 'Alcohol',
#            x = 'Schooling',
#            marginal_y = 'violin',
#            trendline = 'ols')
```

From the above scatter plot we can see that there is some linear relationship between Alcohol to Schooling. Hence, we will impute the missing values of Alcohol regarding to this relationship.

In [19]:
```
imputer = KNNImputer(n_neighbors=1)
arr = imputer.fit_transform(df[['Income composition of resources','Schooling','Life expectancy ']])

Schooling = []

for i in range(len(arr)):
    Schooling.append(arr[i][1])

df['Schooling'] = Schooling
```

Missing Values for BMI & Thinness columns

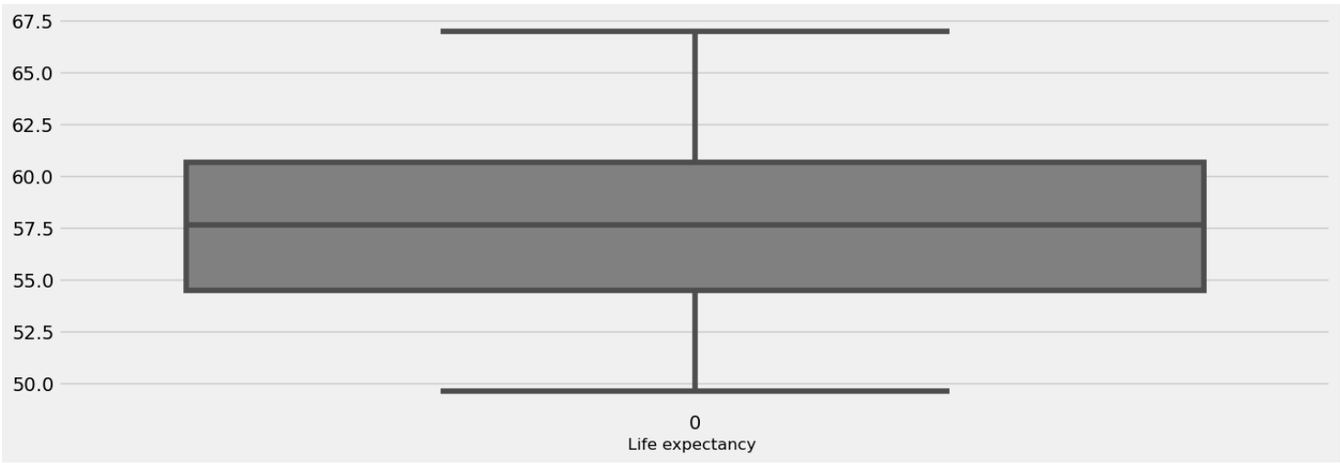
In [20]:
```
newdf = df[df[' BMI '].isnull() &
  df[' thinness 5-9 years'].isnull() &
  df[' thinness  1-19 years'].isnull()]

newdf = newdf.reset_index()
newdf = newdf.drop('index', axis=1)
newdf.head()
```

Out[20]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thin
0	South Sudan	2015	Developing	57.3	332.0	26	0.000000	31.0	878	NaN	39	41.0	NaN	31.0	3.4	758.725782	11882136.0	
1	South Sudan	2014	Developing	56.6	343.0	26	46.074469	NaN	441	NaN	39	44.0	2.74	39.0	3.5	1151.861715	1153971.0	
2	South Sudan	2013	Developing	56.4	345.0	26	47.444530	NaN	525	NaN	40	5.0	2.62	45.0	3.6	1186.113250	1117749.0	
3	South Sudan	2012	Developing	56.0	347.0	26	38.338232	NaN	1952	NaN	40	64.0	2.77	59.0	3.8	958.455810	1818258.0	
4	South Sudan	2011	Developing	55.4	355.0	27	0.000000	NaN	1256	NaN	41	66.0	NaN	61.0	3.9	176.971300	1448857.0	

In [21]:
```
plt.rcParams['figure.figsize'] = (15, 5)
plt.style.use('fivethirtyeight')

# Box plot for Life expectancy

sns.boxplot(newdf['Life expectancy '], color = 'grey')
plt.xlabel('Life expectancy ', fontsize = 12)
print(f"The life expectancy average is: {newdf['Life expectancy '].mean()}, and the standard deviation is: {newdf['Life expectancy '].std()},
```

```
The life expectancy average is: 57.573076923076925, and the standard deviation is: 4.3650940601373245, the min value is: 49.6, and the max is: 67.0
```



From the box plot above we can conclude that for the null values in BMI & thinness columns - the life expectancy almost distributed normally. Hence, we will impute the mean values for the life expectancy that distributed around newdf['Life expectancy'].min() to newdf['Life expectancy'].max()

In [22]:
```
# df[' BMI '] = df[' BMI '].fillna(df[(df['Life expectancy '] >= newdf['Life expectancy '].min()) &
#   (df['Life expectancy '] <= newdf['Life expectancy '].max())][' BMI '].mean())

# df[' thinness 5-9 years'] = df[' thinness 5-9 years'].fillna(df[(df['Life expectancy '] >= newdf['Life expectancy '].min()) &
#   (df['Life expectancy '] <= newdf['Life expectancy '].max())][' thinness 5-9 years'].mean())

# df[' thinness  1-19 years'] = df[' thinness  1-19 years'].fillna(df[(df['Life expectancy '] >= newdf['Life expectancy '].min()) &
#   (df['Life expectancy '] <= newdf['Life expectancy '].max())][' thinness  1-19 years'].mean())
```

In [23]:
```
imputer = KNNImputer(n_neighbors=10)
arr = imputer.fit_transform(df[[' thinness  1-19 years',' thinness 5-9 years',' BMI ','Life expectancy ']])

f = []
s = []
t = []

for i in range(len(arr)):
    f.append(arr[i][0])
    s.append(arr[i][1])
    t.append(arr[i][2])

df[' thinness  1-19 years'] = f
df[' thinness 5-9 years'] = s
df[' BMI '] = t
```

Missing values for Hepatitis B

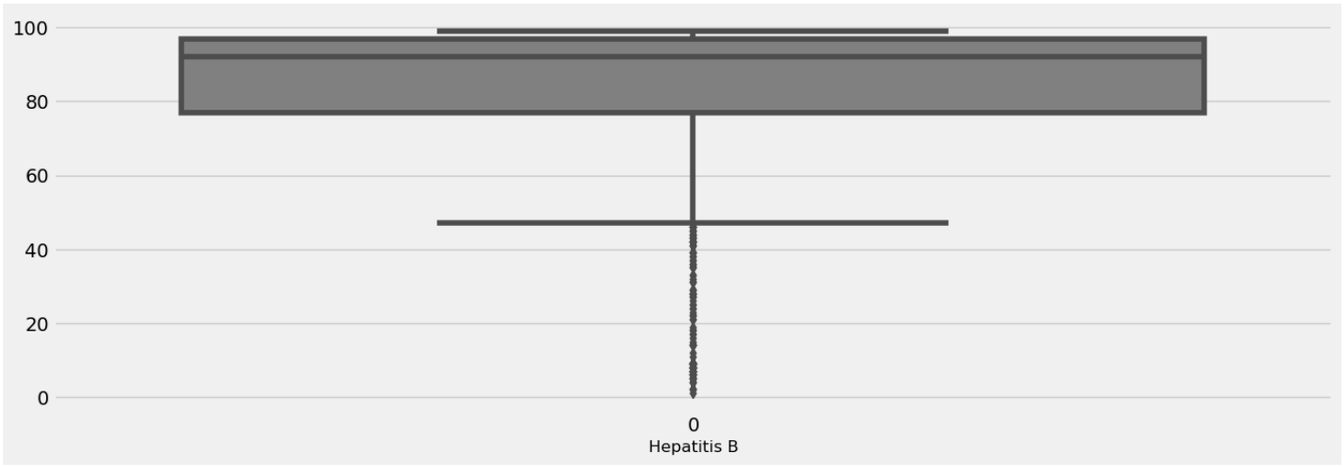
In [24]:
```
df[df['Hepatitis B'].isnull()]
```

Out[24]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP
36	Algeria	2003	Developing	71.7	146.0	20	25.018523	NaN	15374	47.000000	23	87.0	3.60	87.0	0.1	294.335560
37	Algeria	2002	Developing	71.6	145.0	20	148.511984	NaN	5862	46.100000	23	86.0	3.73	86.0	0.1	1774.336730
38	Algeria	2001	Developing	71.4	145.0	20	147.986071	NaN	2686	45.300000	24	89.0	3.84	89.0	0.1	1732.857979
39	Algeria	2000	Developing	71.3	145.0	21	154.455944	NaN	0	44.400000	25	86.0	3.49	86.0	0.1	1757.177970
48	Angola	2006	Developing	47.7	381.0	90	25.086888	NaN	765	18.200000	143	36.0	4.54	34.0	2.5	262.415149
49	Angola	2005	Developing	47.4	382.0	92	98.191451	NaN	258	17.700000	148	39.0	4.10	38.0	2.6	1443.991929
50	Angola	2004	Developing	47.1	386.0	94	8.866777	NaN	29	17.200000	152	4.0	4.71	4.0	2.5	141.868440
51	Angola	2002	Developing	46.5	391.0	96	24.037942	NaN	11945	16.300000	157	37.0	3.63	41.0	2.3	711.181716
52	Angola	2001	Developing	45.7	44.0	97	30.359936	NaN	9046	15.800000	159	41.0	5.38	38.0	2.1	526.168743

In [25]:
```
plt.rcParams['figure.figsize'] = (15, 5)
plt.style.use('fivethirtyeight')

# Box plot for Life expectancy

sns.boxplot(df['Hepatitis B'].values, color = 'grey')
plt.xlabel('Hepatitis B', fontsize = 12)
print(f"The average is: {df['Hepatitis B'].mean()}, and the standard deviation is: {df['Hepatitis B'].std()}, the min value is: {df['Hepatitis B
```

```
The average is: 80.96084210526315, and the standard deviation is: 25.01833664784717, the min value is: 1.0, and the max is: 99.0
```



In [26]:
```
arr = imputer.fit_transform(df[['Hepatitis B','Life expectancy ']])

hep = []

for i in range(len(arr)):
    hep.append(arr[i][0])

df['Hepatitis B'] = hep
```

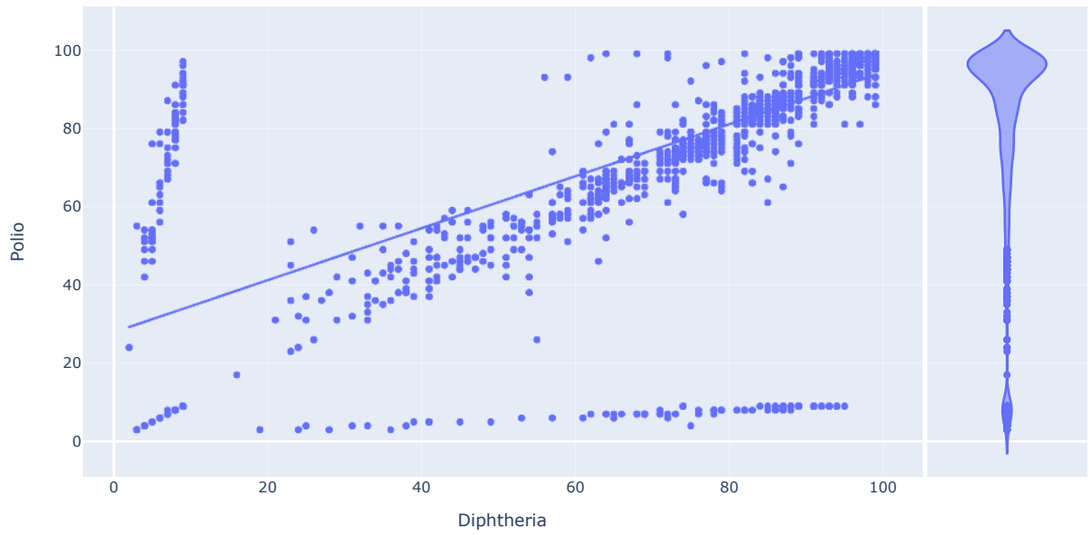
Missing for Polio and Diphtheria

In [27]:
```
df[df['Polio'].isnull()]
```

Out[27]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP
1388	Montenegro	2004	Developing	73.5	134.0	0	57.121901	92.400000	0	55.000000	0	NaN	8.45	NaN	0.1	338.199535
1389	Montenegro	2003	Developing	73.5	134.0	0	495.078296	92.400000	0	54.200000	0	NaN	8.91	NaN	0.1	2789.173500
1390	Montenegro	2002	Developing	73.4	136.0	0	36.480240	71.100000	0	53.500000	0	NaN	8.33	NaN	0.1	216.243274
1391	Montenegro	2001	Developing	73.3	136.0	0	33.669814	94.200000	0	52.700000	0	NaN	8.23	NaN	0.1	199.583957
1935	South Sudan	2010	Developing	55.0	359.0	27	0.000000	72.100000	0	13.240000	41	NaN	NaN	NaN	4.0	1562.239346
1936	South Sudan	2009	Developing	54.3	369.0	27	0.000000	68.200000	0	14.190000	42	NaN	NaN	NaN	4.2	1264.789980
1937	South Sudan	2008	Developing	53.6	377.0	27	0.000000	80.600000	0	20.510000	42	NaN	NaN	NaN	4.2	1678.711862

In [28]:
```
# lets understand the impact of Polio on Diphtheria
px.scatter(df, y = 'Polio',
           x = 'Diphtheria ',
           marginal_y = 'violin',
           trendline = 'ols')
```



From the above correlation map we can see that Polio and Diphtheria are highly correlated. Hence we will impute these missing values regarding their linear relationship.

In [29]:
```
imputer = KNNImputer(n_neighbors=1)
arr = imputer.fit_transform(df[['Polio','Diphtheria ']])

Polio = []
Diphtheria = []

for i in range(len(arr)):
    Polio.append(arr[i][0])
    Diphtheria.append(arr[i][1])

df['Polio'] = Polio
df['Diphtheria '] = Diphtheria
```

Missing Values for Total expenditure

In [30]:
```
df[df['Total expenditure'].isnull()]
```

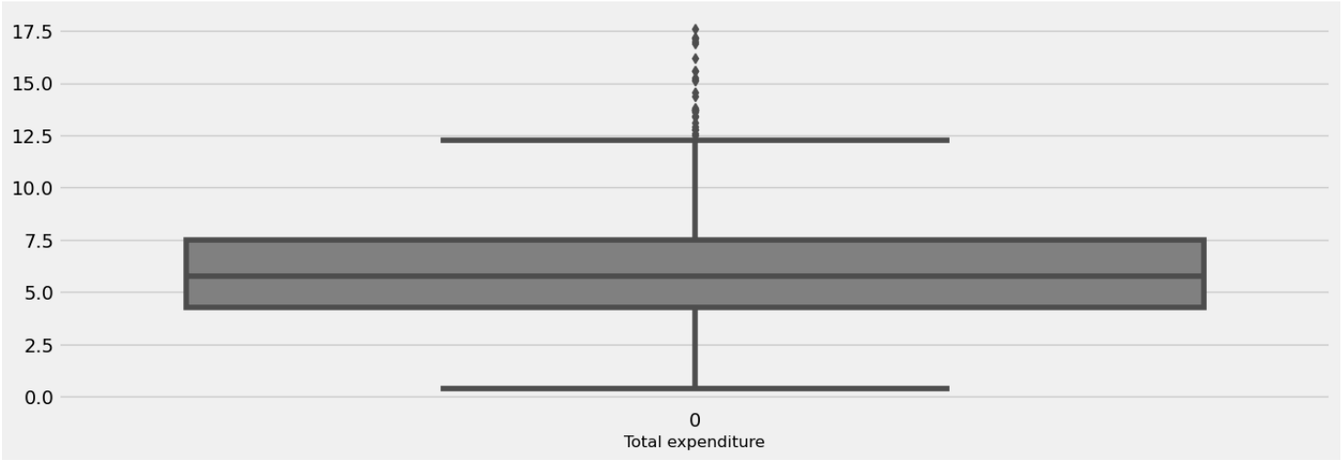
Out[30]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	
40	Angola	2015	Developing	52.4	335.0	66	0.0	64.000000	118	23.300000	98	7.000000	NaN	64.000000	1.9	3695.79
67	Argentina	2015	Developing	76.3	116.0	8	0.0	94.000000	0	62.800000	9	93.000000	NaN	94.000000	0.1	13467.12
89	Australia	2015	Developed	82.8	59.0	1	0.0	93.000000	74	66.600000	1	93.000000	NaN	93.000000	0.1	56554.38
102	Austria	2015	Developed	81.5	65.0	0	0.0	93.000000	309	57.600000	0	93.000000	NaN	93.000000	0.1	43665.94
115	Azerbaijan	2015	Developing	72.7	118.0	5	0.0	96.000000	0	52.500000	6	98.000000	NaN	96.000000	0.1	55.31
124	Bahamas	2015	Developing	76.1	147.0	0	0.0	95.000000	0	64.500000	0	95.000000	NaN	95.000000	0.1	
140	Bahrain	2015	Developing	76.9	69.0	0	0.0	98.000000	0	63.600000	0	98.000000	NaN	98.000000	0.1	22688.87
172	Belarus	2015	Developing	72.3	196.0	0	0.0	99.000000	2	62.300000	0	99.000000	NaN	99.000000	0.1	5949.11
184	Belgium	2015	Developed	81.1	74.0	0	0.0	98.000000	47	63.700000	1	99.000000	NaN	99.000000	0.1	4356.87

We can see that there in no special pattern regarding these null values. Hence, we will impute the mean of this column.

In [31]:
```
plt.rcParams['figure.figsize'] = (15, 5)
plt.style.use('fivethirtyeight')

# Box plot for Life expectancy

sns.boxplot(df['Total expenditure'], color = 'grey')
plt.xlabel('Total expenditure', fontsize = 12)
print(f"The life expectancy average is: {df['Total expenditure'].mean()}, and the standard deviation is: {df['Total expenditure'].std()}, the m
```

```
The life expectancy average is: 5.930162842338996, and the standard deviation is: 2.4832729063507153, the min value is: 0.37, and the max is:
17.6
```



In [32]:
```
df = df.drop(['Total expenditure'], axis = 1)
```

In [33]:
```
# arr = imputer.fit_transform(df[['Total expenditure','Life expectancy ']])

# tot = []

# for i in range(len(arr)):
#     tot.append(arr[i][0])

# df['Total expenditure'] = tot
```

Missing Values for GDP

In [34]:
```
df[df['GDP'].isnull()]
```

Out[34]:

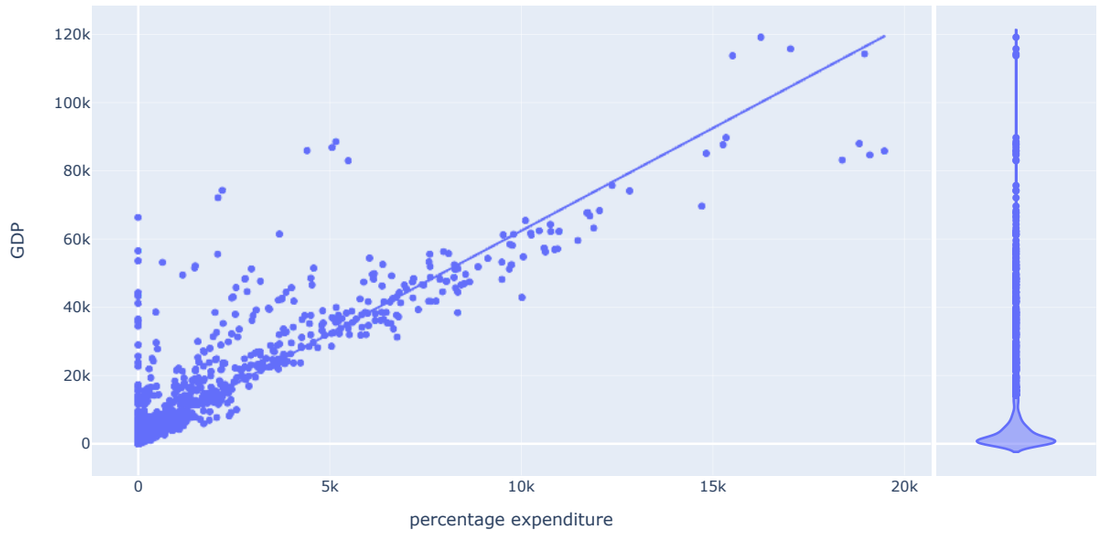
	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thi
124	Bahamas	2015	Developing	76.1	147.0	0	0.0	95.000000	0	64.5	0	95.0	95.0	0.1	NaN	NaN	2.5	
125	Bahamas	2014	Developing	75.4	16.0	0	0.0	96.000000	0	63.8	0	96.0	96.0	0.1	NaN	NaN	2.5	
126	Bahamas	2013	Developing	74.8	172.0	0	0.0	97.000000	0	63.2	0	97.0	97.0	0.1	NaN	NaN	2.5	
127	Bahamas	2012	Developing	74.9	167.0	0	0.0	96.000000	0	62.6	0	99.0	98.0	0.2	NaN	NaN	2.5	
128	Bahamas	2011	Developing	75.0	162.0	0	0.0	95.000000	0	62.0	0	97.0	98.0	0.1	NaN	NaN	2.5	
129	Bahamas	2010	Developing	75.0	161.0	0	0.0	98.000000	0	61.3	0	97.0	99.0	0.2	NaN	NaN	2.5	
130	Bahamas	2009	Developing	74.6	168.0	0	0.0	95.000000	0	6.7	0	97.0	96.0	0.1	NaN	NaN	2.5	
131	Bahamas	2008	Developing	74.5	167.0	0	0.0	9.000000	0	6.1	0	93.0	93.0	0.1	NaN	NaN	2.5	
132	Bahamas	2007	Developing	74.4	167.0	0	0.0	93.000000	0	59.4	0	95.0	95.0	0.1	NaN	NaN	2.5	

In [35]:
```
df[['GDP','percentage expenditure']].corr()
```

Out[35]:

	GDP	percentage expenditure
GDP	1.00000	0.89937
percentage expenditure	0.89937	1.00000

In [36]:
```
# the impact of Alcohol on Population
px.scatter(df, y = 'GDP',
           x = 'percentage expenditure',
           marginal_y = 'violin',
           trendline = 'ols')
```



From the Fig. above we can conclude that 'GDP' & 'percentage expenditure' share a good linear relationship which we can impute this data from their linear relationship without considering the whole data.

- We can also conclude that there are many outliers in these features which might hurt our model, we will treat outliers and anomalies further in our work.

In [37]:
```
imputer = KNNImputer(n_neighbors=1)
arr = imputer.fit_transform(df[['GDP','percentage expenditure']])

GDP = []

for i in range(len(arr)):
    GDP.append(arr[i][0])

df['GDP'] = GDP
```

Missing values for Population

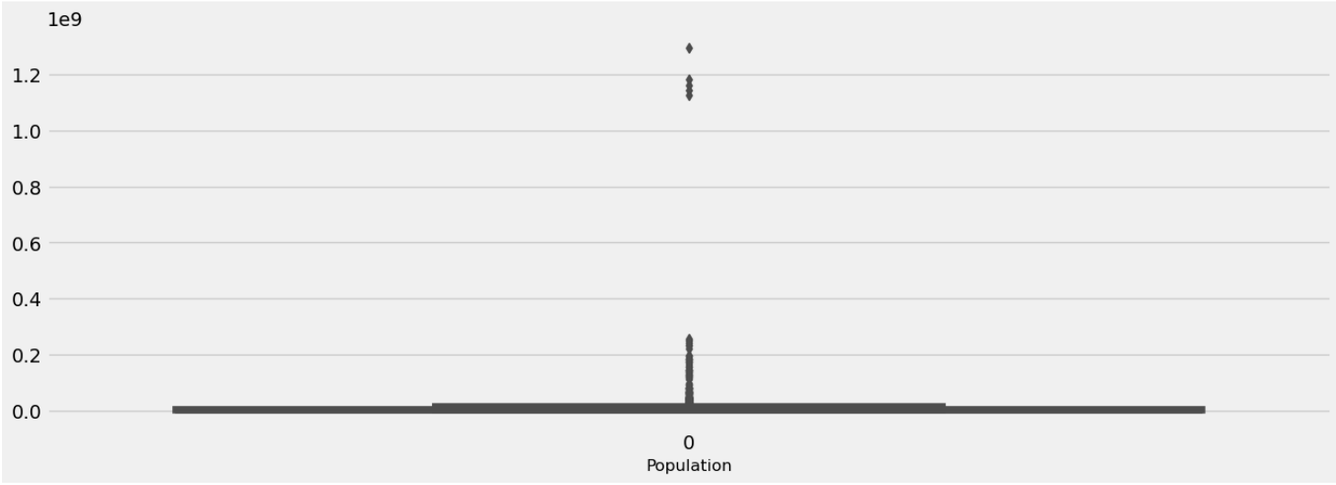
In [38]:
```
df[df['Population'].isnull()]
```

Out[38]:

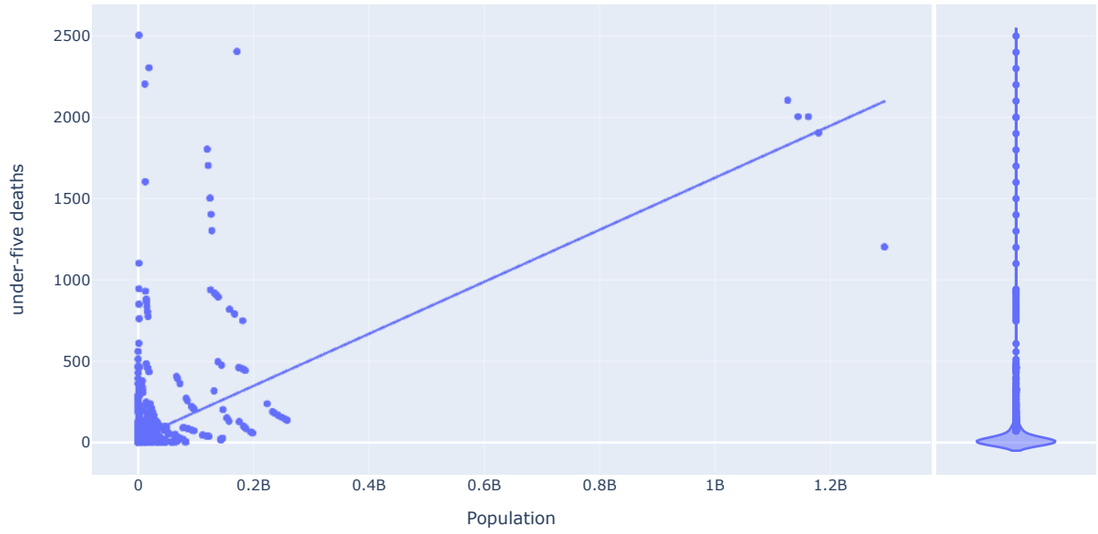
	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	GDP	Population	thinne 1- yea
54	Antigua and Barbuda	2014	Developing	76.2	131.0	0	2422.999774	99.000000	0	47.0	0	96.0	99.0	0.2	12888.296670	NaN	3
55	Antigua and Barbuda	2013	Developing	76.1	133.0	0	1991.430372	99.000000	0	46.4	0	98.0	99.0	0.2	12224.864160	NaN	3
56	Antigua and Barbuda	2012	Developing	75.9	134.0	0	2156.229842	98.000000	0	45.7	0	97.0	98.0	0.2	12565.441970	NaN	3
57	Antigua and Barbuda	2011	Developing	75.7	136.0	0	1810.875316	99.000000	0	45.1	0	99.0	99.0	0.1	11929.349910	NaN	3
	Antigua																

In [39]:
```
plt.rcParams['figure.figsize'] = (15, 5)
plt.style.use('fivethirtyeight')

# Box plot for Life expectancy

sns.boxplot(df['Population'], color = 'grey')
plt.xlabel('Population', fontsize = 12)
print(f"The life expectancy average is: {df['Population'].mean()}, and the standard deviation is: {df['Population'].std()}, the min value is:
```

```
The life expectancy average is: 12764541.774711037, and the standard deviation is: 61037647.16463728, the min value is: 34.0, and the max is:
1293859294.0
```



In [40]:
```
# the impact of Alcohol on Population
px.scatter(df, y = 'under-five deaths ',
           x = 'Population',
           marginal_y = 'violin',
           trendline = 'ols')
```



From the Fig. above we can conclude that 'under-five deaths' & 'Population' share some linear relationship which we can impute this data from their linear relationship without considering the whole data.

- We can also conclude that there are many outliers in these features which might hurt our model, we will treat outliers and anomalies further in our work.

In [41]:
```
df = df.drop(['Population'], axis = 1)
```

In [42]:
```
# imputer = KNNImputer(n_neighbors=1)
# arr = imputer.fit_transform(df[['under-five deaths ','Population']])

# Population = []

# for i in range(len(arr)):
#     Population.append(arr[i][1])

# df['Population'] = Population
```

Missing Values for Income composition of resources

In [43]:
```
df[df['Income composition of resources'].isnull()]
```

Out[43]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	GDP	thinness 1-19 years	thinness 5-9 years	co
344	Côte d'Ivoire	2012	Developing	52.0	415.0	59	0.0	82.000000	137	26.2	82	83.0	82.0	2.9	3695.793748	5.9	5.9	
345	Côte d'Ivoire	2011	Developing	51.7	419.0	60	0.0	62.000000	628	25.6	83	58.0	62.0	3.3	3695.793748	6.1	6.0	
346	Côte d'Ivoire	2010	Developing	51.5	417.0	60	0.0	85.000000	441	25.0	84	81.0	85.0	3.3	3695.793748	6.3	6.2	
347	Côte d'Ivoire	2009	Developing	51.0	426.0	60	0.0	81.000000	183	24.4	84	77.0	81.0	3.7	3695.793748	6.5	6.4	
348	Côte d'Ivoire	2008	Developing	54.0	437.0	60	0.0	74.000000	12	23.8	85	58.0	74.0	4.1	3695.793748	6.6	6.6	
349	Côte d'Ivoire	2007	Developing	49.9	443.0	61	0.0	76.000000	5	23.2	87	75.0	76.0	5.3	3695.793748	6.8	6.7	
350	Côte d'Ivoire	2005	Developing	48.7	466.0	63	0.0	76.000000	115	22.1	90	87.0	76.0	6.1	3695.793748	7.2	7.1	
351	Côte d'Ivoire	2004	Developing	48.2	472.0	64	0.0	67.000000	3466	21.5	91	76.0	67.0	6.5	3695.793748	7.3	7.3	
352	Côte d'Ivoire	2003	Developing	48.0	473.0	64	0.0	63.000000	4770	2.9	92	68.0	61.0	6.7	3695.793748	7.5	7.5	
353	Côte d'Ivoire	2002	Developing	47.7	473.0	65	0.0	48.000000	5882	2.4	93	7.0	64.0	6.9	3695.793748	7.7	7.7	
354	Côte d'Ivoire	2001	Developing	47.8	467.0	65	0.0	1.000000	5790	19.9	94	7.0	66.0	7.0	3695.793748	7.9	7.9	
355	Côte d'Ivoire	2000	Developing	47.9	461.0	67	0.0	70.600000	5729	19.4	95	66.0	65.0	7.1	3695.793748	8.1	8.1	
545	Czechia	2015	Developed	78.8	86.0	0	0.0	97.000000	9	66.1	0	97.0	97.0	0.1	3695.793748	1.8	1.8	
546	Czechia	2014	Developed	78.6	88.0	0	0.0	99.000000	222	65.6	0	99.0	97.0	0.1	3695.793748	1.8	1.9	
547	Czechia	2013	Developed	78.2	9.0	0	0.0	99.000000	15	65.1	0	99.0	98.0	0.1	3695.793748	1.8	1.9	
548	Czechia	2011	Developed	77.8	97.0	0	0.0	99.000000	17	64.0	0	99.0	99.0	0.1	3695.793748	1.9	1.9	
549	Czechia	2010	Developed	77.5	99.0	0	0.0	99.000000	0	63.6	0	99.0	99.0	0.1	3695.793748	1.9	2.0	
550	Czechia	2009	Developed	77.1	12.0	0	0.0	99.000000	5	63.1	0	99.0	99.0	0.1	3695.793748	1.9	2.0	
551	Czechia	2008	Developed	77.0	16.0	0	0.0	99.000000	2	62.6	0	99.0	99.0	0.1	3695.793748	1.9	2.0	
552	Czechia	2007	Developed	76.8	17.0	0	0.0	99.000000	2	62.2	0	99.0	99.0	0.1	3695.793748	2.0	2.1	
553	Czechia	2006	Developed	76.5	19.0	0	0.0	98.000000	7	61.8	0	98.0	98.0	0.1	3695.793748	2.0	2.1	
554	Czechia	2004	Developed	75.8	116.0	0	0.0	98.000000	17	6.8	0	96.0	98.0	0.1	3695.793748	2.1	2.2	
555	Czechia	2003	Developed	75.2	122.0	0	0.0	92.000000	30	6.4	0	97.0	97.0	0.1	3695.793748	2.2	2.2	
556	Czechia	2002	Developed	75.3	12.0	0	0.0	86.000000	4	59.9	0	97.0	98.0	0.1	3695.793748	2.2	2.3	
557	Czechia	2001	Developed	75.1	123.0	0	0.0	95.100000	6	59.4	0	97.0	98.0	0.1	3695.793748	2.2	2.3	
558	Czechia	2000	Developed	74.7	126.0	0	0.0	83.400000	9	59.0	0	98.0	98.0	0.1	3695.793748	2.3	2.3	
559	Democratic People's Republic of Korea	2015	Developing	76.0	139.0	6	0.0	96.000000	0	32.9	7	99.0	96.0	0.1	3695.793748	4.9	4.9	
560	Democratic People's Republic of Korea	2014	Developing	73.0	142.0	6	0.0	93.000000	3	32.4	8	99.0	93.0	0.1	3695.793748	4.9	4.9	
561	Democratic People's Republic of Korea	2013	Developing	71.0	146.0	6	0.0	93.000000	0	31.8	8	99.0	93.0	0.1	3695.793748	5.0	5.0	
562	Democratic People's Republic of Korea	2012	Developing	69.8	149.0	7	0.0	96.000000	0	31.3	9	99.0	96.0	0.1	3695.793748	5.1	5.1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2478	Democratic People's Republic of Korea	2010	Developing	NaN	157.0	8	0.0	93.000000	0	3.3	10	99.0	93.0	0.1	3695.793748	5.2	5.2	
2479	Democratic Republic of the Congo	2013	Developing	NaN	272.0	238	0.0	74.000000	88381	2.6	314	74.0	74.0	1.2	3695.793748	9.9	9.6	
2480	Democratic Republic of the Congo	2005	Developing	NaN	314.0	233	0.0	80.960842	182485	16.9	329	6.0	6.0	2.0	3695.793748	11.5	11.3	
2481	Democratic Republic of the Congo	2004	Developing	NaN	323.0	232	0.0	80.960842	44934	16.5	330	52.0	54.0	2.1	3695.793748	11.7	11.5	
2482	Democratic Republic of the Congo	2002	Developing	NaN	341.0	229	0.0	80.960842	30466	15.7	331	4.0	38.0	2.4	3695.793748	12.1	11.9	
2746	Republic of Korea	2013	Developing	NaN	68.0	1	0.0	99.000000	107	3.8	2	99.0	99.0	0.1	3695.793748	1.5	1.0	
2747	Republic of Korea	2010	Developing	NaN	74.0	2	0.0	94.000000	114	29.5	2	95.0	94.0	0.1	3695.793748	1.5	1.0	
2748	Republic of Korea	2006	Developing	NaN	79.0	2	0.0	99.000000	28	27.7	2	98.0	98.0	0.1	3695.793748	1.5	1.0	
2749	Republic of Moldova	2006	Developing	NaN	242.0	1	0.0	98.000000	34	48.7	1	98.0	97.0	0.1	3695.793748	3.1	3.3	
2750	Republic of Moldova	2002	Developing	NaN	225.0	1	0.0	99.000000	4929	47.2	1	98.0	97.0	0.1	3695.793748	3.4	3.6	
2796	Somalia	2013	Developing	NaN	318.0	51	0.0	42.000000	3173	23.3	81	47.0	42.0	0.8	47.543235	6.8	6.6	
2797	Somalia	2010	Developing	NaN	336.0	52	0.0	80.960842	115	22.0	83	49.0	45.0	0.8	3695.793748	7.0	6.8	
2884	United Kingdom of Great Britain and Northern I...	2014	Developed	NaN	71.0	3	0.0	80.960842	133	66.0	4	95.0	95.0	0.1	3695.793748	0.8	0.5	
2885	United Kingdom of Great Britain and Northern I...	2013	Developed	NaN	72.0	3	0.0	80.960842	1919	65.4	4	95.0	95.0	0.1	3695.793748	0.8	0.5	

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	GDP	thinness 1-19 years	thinness 5-9 years	co
2886	United Kingdom of Great Britain and Northern I...	2005	Developed	NaN	82.0	4	0.0	80.960842	79	6.7	4	91.0	91.0	0.1	3695.793748	0.7	0.5	
2887	United Kingdom of Great Britain and Northern I...	2003	Developed	NaN	86.0	4	0.0	80.960842	460	59.5	4	91.0	91.0	0.1	3695.793748	0.7	0.5	
2888	United Republic of Tanzania	2014	Developing	NaN	34.0	86	0.0	97.000000	88	23.2	121	97.0	97.0	1.4	3695.793748	6.8	6.6	
2889	United Republic of Tanzania	2009	Developing	NaN	368.0	91	0.0	85.000000	1574	2.2	135	88.0	85.0	6.4	3695.793748	7.2	7.2	
2890	United Republic of Tanzania	2007	Developing	NaN	411.0	94	0.0	83.000000	7726	19.1	141	88.0	83.0	8.5	3695.793748	7.5	7.4	
2891	United Republic of Tanzania	2004	Developing	NaN	454.0	100	0.0	95.000000	1419	17.7	154	95.0	95.0	10.8	3695.793748	7.8	7.8	
2892	United Republic of Tanzania	2001	Developing	NaN	46.0	110	0.0	80.960842	11847	16.4	173	65.0	87.0	12.5	3695.793748	8.2	8.1	
2893	United Republic of Tanzania	2000	Developing	NaN	457.0	114	0.0	80.960842	14649	16.0	181	64.0	79.0	12.8	3695.793748	8.3	8.3	
2894	United States of America	2014	Developed	NaN	14.0	23	0.0	92.000000	667	69.1	27	93.0	95.0	0.1	3695.793748	0.8	0.6	
2895	United States of America	2013	Developed	NaN	16.0	23	0.0	91.000000	187	68.6	27	93.0	94.0	0.1	3695.793748	0.7	0.6	
2896	United States of America	2010	Developed	NaN	15.0	25	0.0	92.000000	63	66.9	30	93.0	95.0	0.1	3695.793748	0.7	0.6	
2897	United States of America	2009	Developed	NaN	18.0	26	0.0	92.000000	71	66.3	31	93.0	95.0	0.1	3695.793748	0.7	0.6	
2898	United States of America	2007	Developed	NaN	11.0	27	0.0	93.000000	43	65.1	32	93.0	96.0	0.1	3695.793748	0.7	0.6	
2899	United States of America	2003	Developed	NaN	114.0	28	0.0	92.000000	56	62.4	33	91.0	96.0	0.1	3695.793748	0.7	0.6	
2900	United States of America	2002	Developed	NaN	115.0	28	0.0	88.000000	41	61.7	33	9.0	94.0	0.1	3695.793748	0.8	0.6	
2901	United States of America	2000	Developed	NaN	114.0	28	0.0	9.000000	85	6.1	33	9.0	94.0	0.1	3695.793748	0.8	0.7	

160 rows × 20 columns

In [44]:
```
df[['Life expectancy ','Income composition of resources','Schooling']].corr()
```

Out[44]:

	Life expectancy	Income composition of resources	Schooling
Life expectancy	1.000000	0.730757	0.757089
Income composition of resources	0.730757	1.000000	0.800046
Schooling	0.757089	0.800046	1.000000

In [45]:
```
# px.scatter_3d(df.sort_values(by='Year'),
#               y='Income composition of resources',x='Life expectancy ',z='Schooling',size='Life expectancy ',
#               template='plotly_dark', color='Status')
```

From the above 3D Scatter plot we can clearly see that there is multivariate linear relationship between these features. Hence we will impute the missing values of 'Income composition of resources' regarding the nearest point for each missing value.

- We can also see that Developed countries have much more Income, life expectancy and schooling.

In [46]:
```
imputer = KNNImputer(n_neighbors=1)
arr = imputer.fit_transform(df[['Life expectancy ','Income composition of resources','Schooling']])

Income = []

for i in range(len(arr)):
    Income.append(arr[i][1])

df['Income composition of resources'] = Income
```

Feature Engineering & Outliers Detection

Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms. Feature engineering can be considered as applied machine learning itself.

- There are mutliple ways of performing feature engineering.
- So many people in the Industry consider it the most important step to improve the Model Performance.
- We should always understand the columns well to make some new features using the old existing features.
- Let's discuss the ways how we can perform feature engineering
  - We can perform Feature Engineering by Removing Unnecassary Columns
  - We can do it by Extracting Features from the Date and Time Features.
  - We can do it by Extracting Features from the Categorcial Features.
  - We can do it by Binnning the Numerical and Categorical Features.
  - We can do it by Aggregating Multiple Features together by using simple Arithmetic operations
- Here, we are only going to perform Feature Engineering by Aggregating some features together.

The presence of outliers in a classification or regression dataset can result in a poor fit and lower predictive modeling performance. Instead, automatic outlier detection methods can be used in the modeling pipeline and compared, just like other data preparation transforms that may be applied to the dataset.

In [47]:
```
# Lets first analyze the Numberical Columns
df.select_dtypes('number').head()
```

Out[47]:

	Year	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	GDP	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling	ID
0	2015	65.0	263.0	62	71.279624	65.0	1154	19.1	83	6.0	65.0	0.1	584.259210	17.2	17.3	0.479	10.1	NaN
1	2013	59.9	268.0	66	73.219243	64.0	430	18.1	89	62.0	64.0	0.1	631.744976	17.7	17.7	0.470	9.9	NaN
2	2012	59.5	272.0	69	78.184215	67.0	2787	17.6	93	67.0	67.0	0.1	669.959000	17.9	18.0	0.463	9.8	NaN
3	2011	59.2	275.0	71	7.097109	68.0	3013	17.2	97	68.0	68.0	0.1	63.537231	18.2	18.2	0.454	9.5	NaN
4	2010	58.8	279.0	74	79.679367	66.0	1989	16.7	102	66.0	66.0	0.1	553.328940	18.4	18.4	0.448	9.2	NaN

In [48]:
```
df.describe()
```

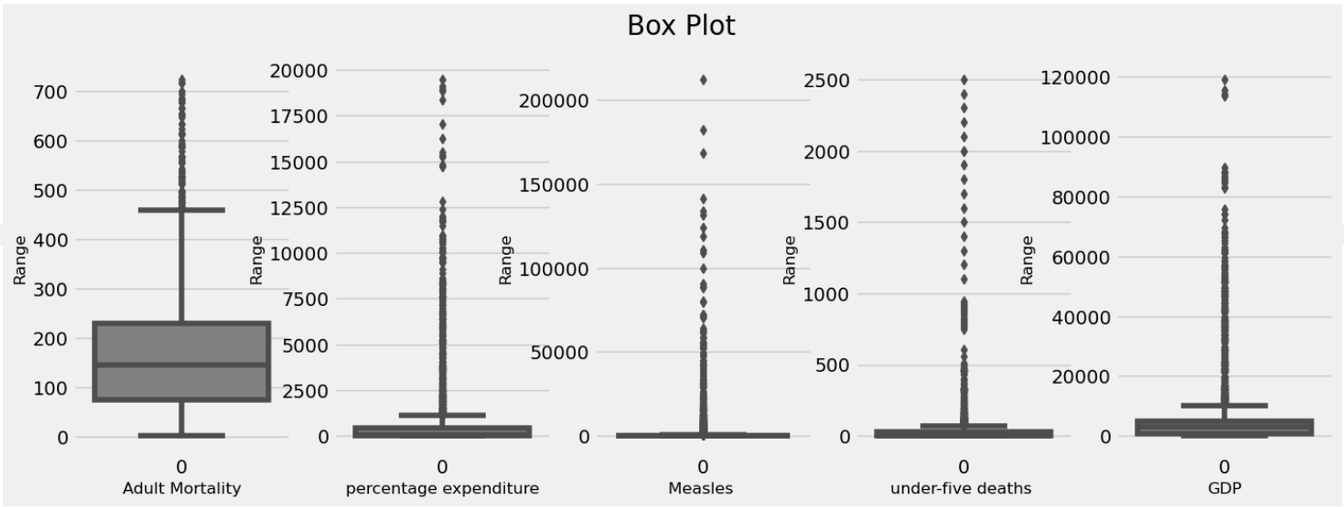
Out[48]:

	Year	Life expectancy	Adult Mortality	infant deaths	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AIDS	
count	2928.00000	2333.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.000000	2928.00
mean	2007.50000	69.302443	164.796448	30.407445	740.321185	80.502571	2427.855874	38.072299	42.179303	82.548298	82.321416	1.747712	6919.51
std	4.61056	9.514162	124.292079	118.114450	1990.930605	22.948787	11485.970937	19.930961	160.700547	23.340548	23.629576	5.085542	13227.36
min	2000.00000	39.000000	1.000000	0.000000	0.000000	1.000000	0.000000	1.000000	0.000000	3.000000	2.000000	0.100000	1.68
25%	2003.75000	63.500000	74.000000	0.000000	4.853964	75.000000	0.000000	19.200000	0.000000	78.000000	78.000000	0.100000	578.79
50%	2007.50000	72.100000	144.000000	3.000000	65.611455	89.000000	17.000000	43.000000	4.000000	93.000000	93.000000	0.100000	3112.56
75%	2011.25000	75.800000	228.000000	22.000000	442.614322	96.000000	362.250000	56.100000	28.000000	97.000000	97.000000	0.800000	4793.63
max	2015.00000	89.000000	723.000000	1800.000000	19479.911610	99.000000	212183.000000	77.600000	2500.000000	99.000000	99.000000	50.600000	119172.74

In [49]:
```
df.columns
```

Out[49]:
```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'percentage expenditure', 'Hepatitis B', 'Measles ',
       ' BMI ', 'under-five deaths ', 'Polio', 'Diphtheria ', ' HIV/AIDS',
       'GDP', ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling', 'ID'],
      dtype='object')
```

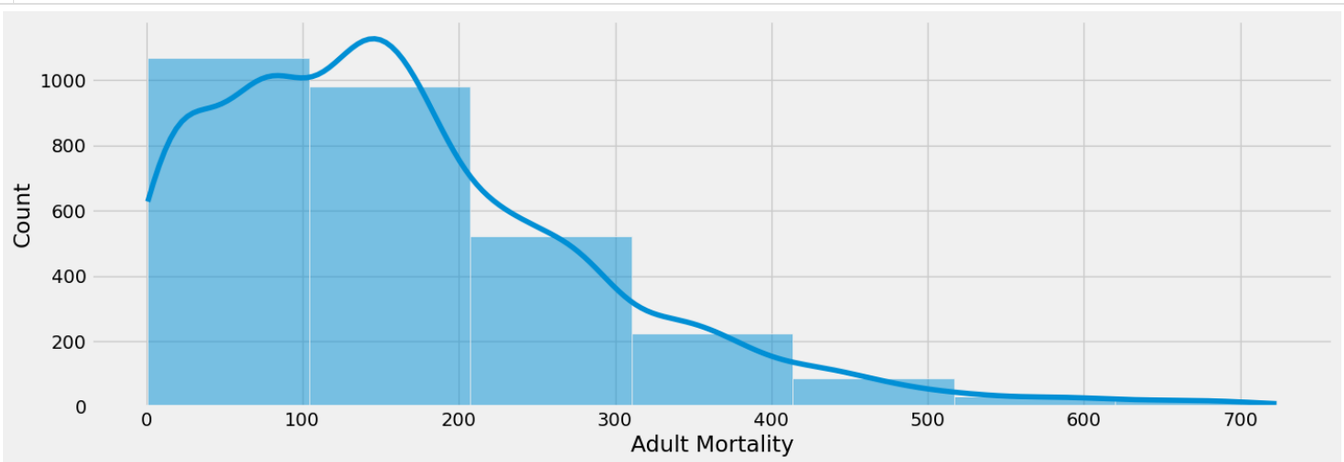
In [50]:
```
# check the boxplots for the columns where we suspect for outliers↔
```



These are the columns that we suspect that we have outliers that can cause a missprediction to our model. We are now going to treat each one of them seperatly.

Outlier Detection for Adult Mortality

In [51]:
```
sns.histplot(x='Adult Mortality', data=df, bins=7,kde=True);
```



In [52]:
```
print(len(df[df['Adult Mortality'] > 600])/len(df))
```

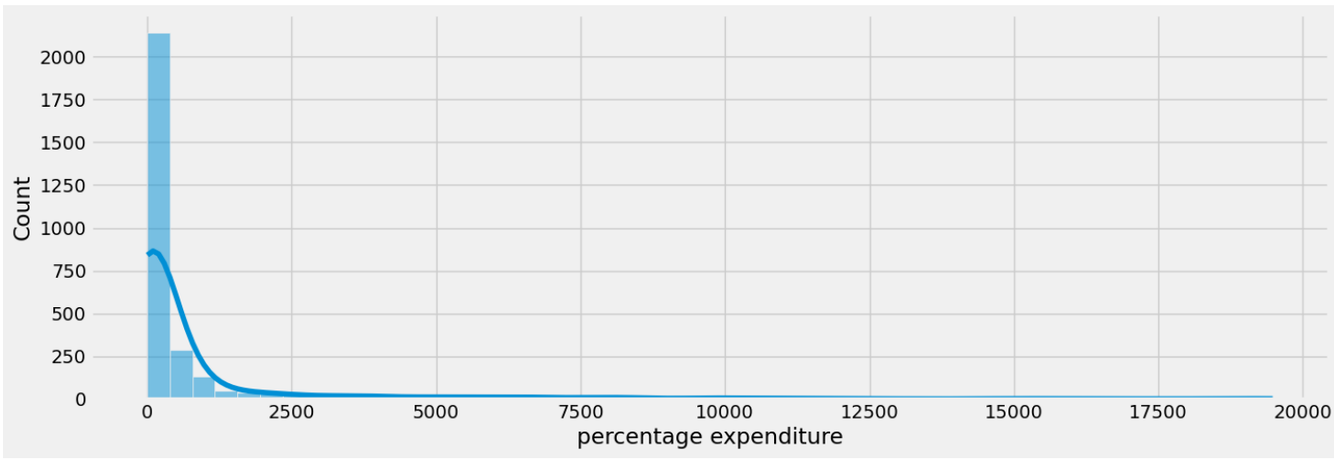
```
0.007513661202185792
```

The percentage of Adult Mortality that their values is more than 500 is less then 1%, Hence we can drop the values that higher which they consider as outliers.

In [53]:
```
df = df[df['Adult Mortality'] < 700]
```

Outlier Detection for percentage expenditure

In [54]:
```
sns.histplot(x='percentage expenditure', data=df, bins=50,kde=True);
```



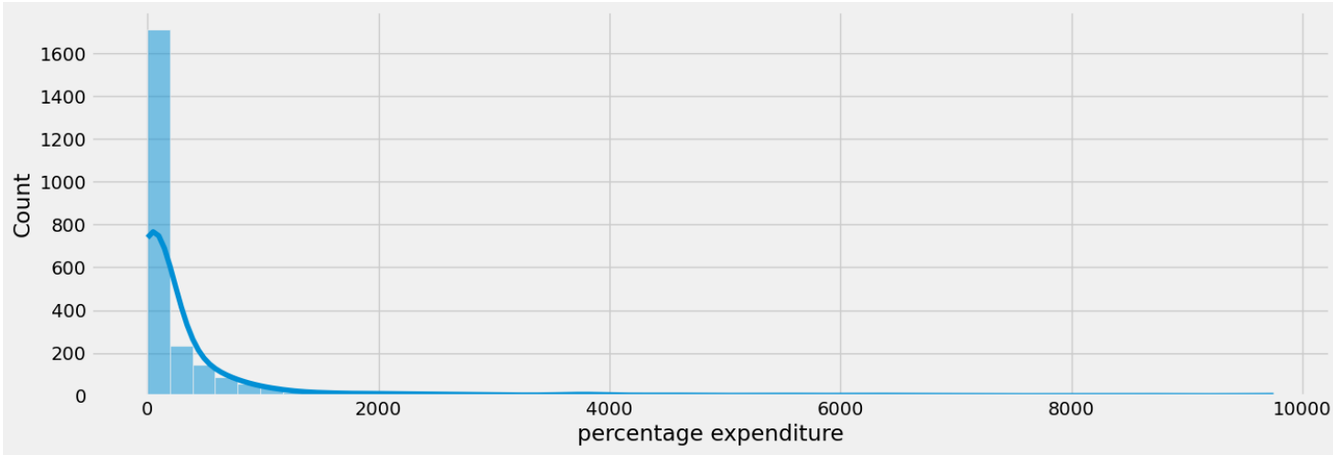
In [55]:
```
len(df[(df['percentage expenditure'] > 5000) &
  (df['Status'] == 'Developed')])/len(df[df['percentage expenditure'] > 5000])
```

Out[55]: 0.8455882352941176

The percentage of percentage expenditure that is higher than 5000 and from developed countries is 86% from all the percentage expenditure that are higher than 5000.

Let's check the distribution of the percentage expenditure around developing countries.

In [56]:
```
sns.histplot(x='percentage expenditure', data=df[(df['Status'] == 'Developing')], bins=50,kde=True);
```



In [57]:
```
len(df[(df['percentage expenditure'] > 3000) &
  (df['Status'] == 'Developing')])/len(df[(df['Status'] == 'Developing')])
```

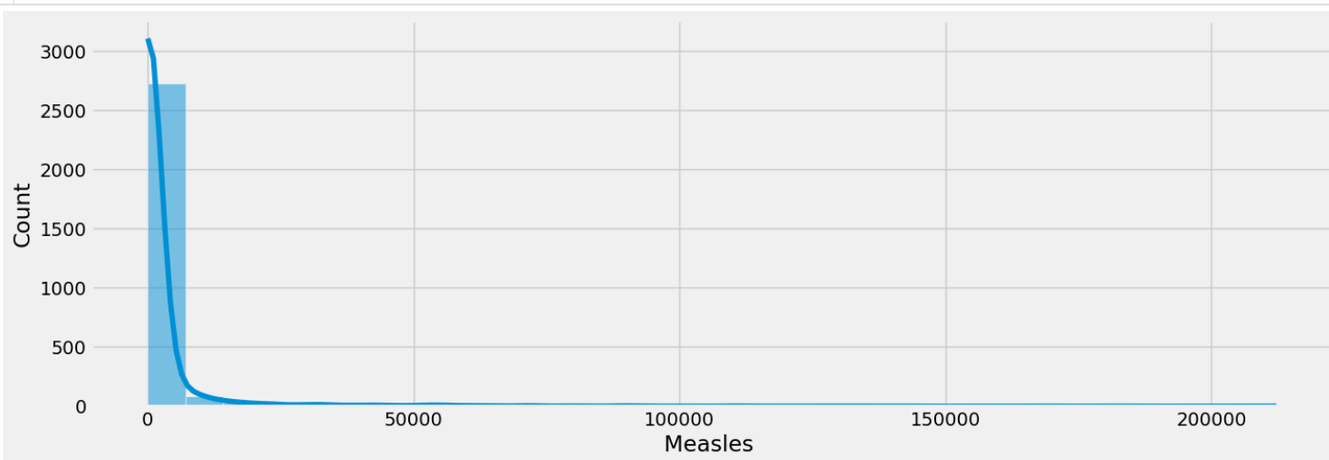
Out[57]: 0.02237878159966846

The percent of the percentage expenditure that is higher than 2000 and from all developing countries is 2.2% from all the data. Hence, we will drop these columns.

In [58]:
```
# df.drop(df[(df['percentage expenditure'] > 3000) &
#   (df['Status'] == 'Developing')].index, inplace=True)
```

Outlier Detection for Measles

In [59]:
```
sns.histplot(x='Measles ', data=df, bins=30,kde=True);
```



In [60]:
```
round(len(df[df['Measles '] > 25000])/len(df),3)
```

Out[60]: 0.023

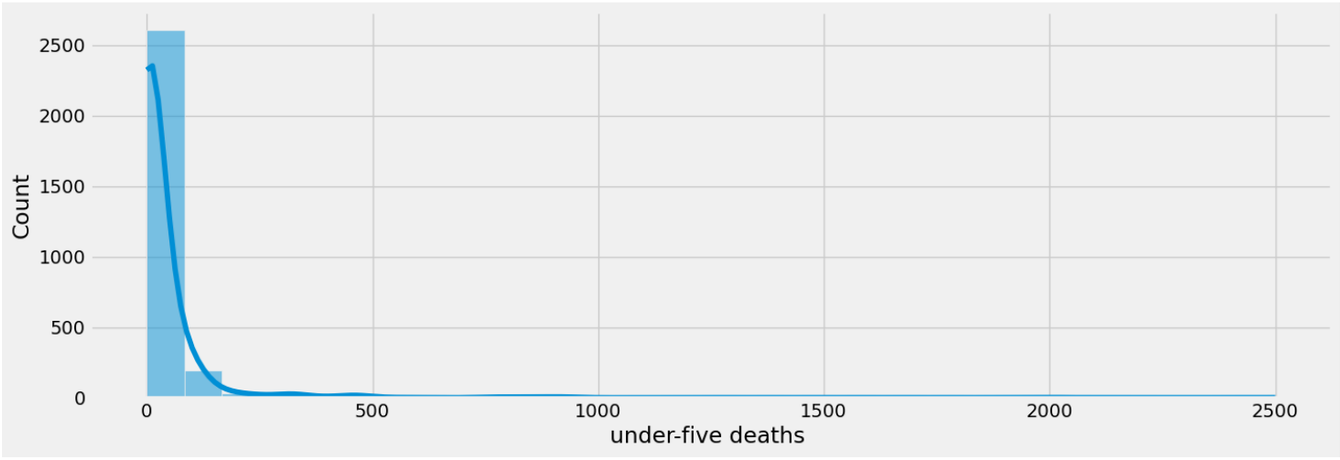
The percentage of Measles that is > 25000 and from all of the data is 2.6% from all the data. Hence, we will drop these columns.

In [61]:
```
# df = df[df['Measles '] < 25000]
```

In [62]:
```
df = df.drop(['Measles '], axis = 1)
```

Outlier Detection for under-five deaths

In [63]:
```
sns.histplot(x='under-five deaths ', data=df, bins=30,kde=True);
```



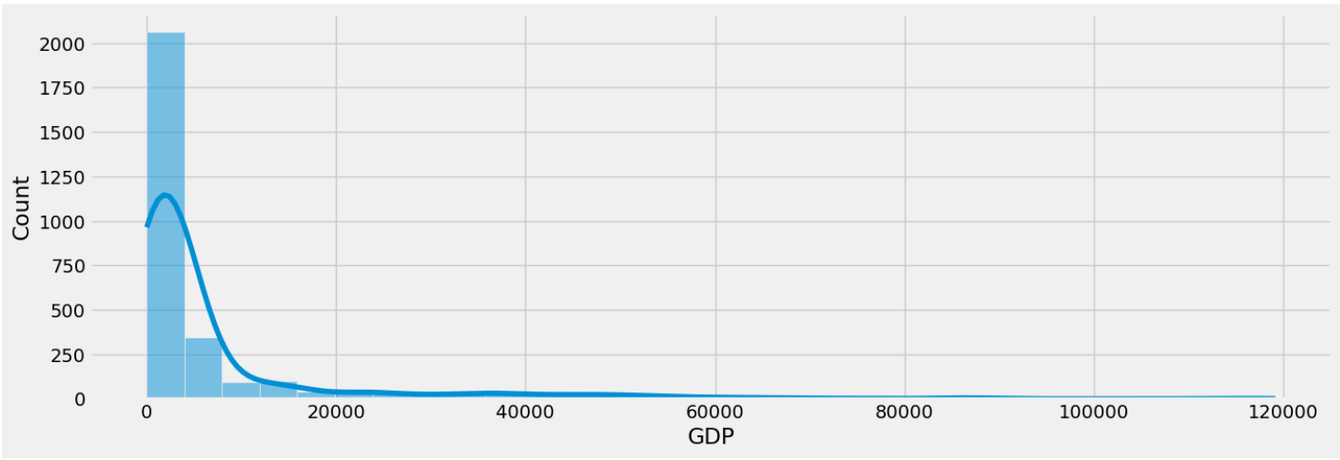
In [64]:
```
round(len(df[df['under-five deaths '] > 300])/len(df),3)
```

Out[64]: 0.028

In [65]:
```
# df = df[df['under-five deaths '] < 300]
```

Outlier Detection for GDP

In [66]:
```
sns.histplot(x='GDP', data=df, bins=30,kde=True);
```



In [67]:
```
round(len(df[df['GDP'] > 20000])/len(df),3)
```

Out[67]: 0.096

In [68]:
```
# df = df[df['GDP'] < 20000]
```

In [69]:
```
df.columns
```

Out[69]:
```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'percentage expenditure', 'Hepatitis B', ' BMI ',
       'under-five deaths ', 'Polio', 'Diphtheria ', ' HIV/AIDS', 'GDP',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling', 'ID'],
      dtype='object')
```

In [70]:
```
test.columns
```

Out[70]:
```
Index(['ID', 'Country', 'Year', 'Status', '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'],
      dtype='object')
```

In [71]:
```
min(df['under-five deaths '])
```

Out[71]: 0

In [72]:
```
min(test['under-five deaths '])
```

Out[72]: 0

In [73]:
```
for i in df.columns:
    if i != 'Life expectancy ':
        print(i)
        print('min df')
        print(min(df[i]))
        print('min test')
        print(min(test[i]))
        print('max df')
        print(max(df[i]))
        print('max test')
        print(max(test[i]))
```

```
27.7
 thinness 5-9 years
min df
0.1
min test
0.1
max df
28.6
max test
28.6
Income composition of resources
min df
0.0
min test
0.0
max df
0.948
max test
0.942
```

In [ ]:

In [74]:
```
difference = set(df['Country'].values).difference(set(test['Country'].values))

l = list(difference)

df = df[~df['Country'].isin(l)]
```

In [ ]:

In [ ]:

In [ ]:

Univariate analysis

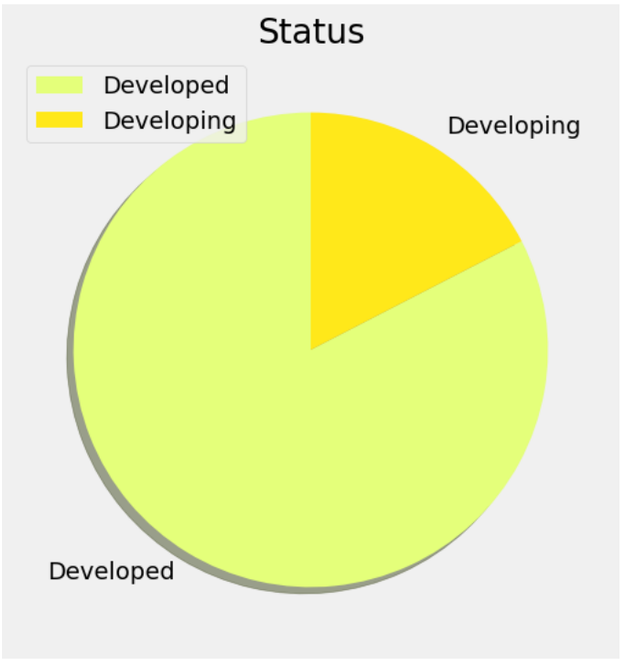
Univariate Analysisis perhaps the simplest form of statistical analysis. Like other forms of statistics, it can be inferential or descriptive. The key fact is that only one variable is involved. Univariate analysis can yield misleading results in cases in which multivariate analysis is more appropriate.

- This is an Essential step, to understand the variables present in the dataset one by one.
- First, we will check the Univariate Analysis for Numerical Columns to check for Outliers by using Box plots.
- Then, we will use Distribution plots to check the distribution of the Numerical Columns in the Dataset.
- After that we will check the Univariate Analysis for Categorical Columns using Pie charts, and Count plots.
- We Use Pie charts, when we have very few categories in the categorical column, and we use count plots we have more categorises in the dataset.

In [75]:
```
# Check the length unqiue values of each column
for column in list(df.columns):
    print(column,':' ,len(set(list(df[column]))))
```

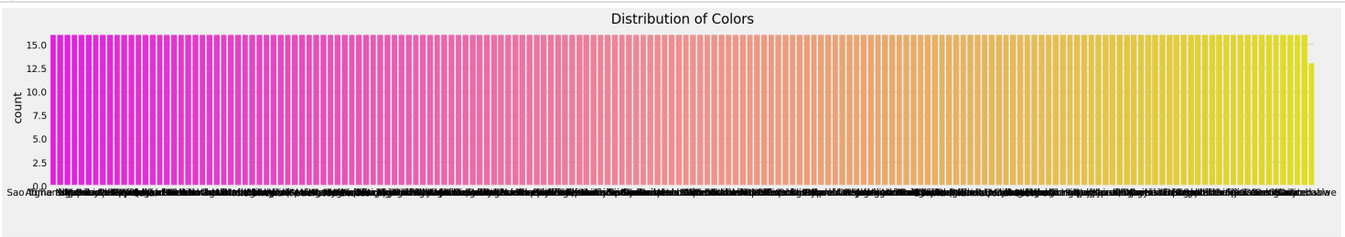
```
Country : 178
Year : 16
Status : 2
Life expectancy  : 953
Adult Mortality : 422
infant deaths : 209
percentage expenditure : 2265
Hepatitis B : 243
 BMI  : 627
under-five deaths  : 252
Polio : 74
Diphtheria  : 82
 HIV/AIDS : 197
GDP : 2422
 thinness  1-19 years : 225
 thinness 5-9 years : 231
Income composition of resources : 625
Schooling : 174
ID : 2845
```

In [76]:
```
# plot pie chart for the columns where we have very few categories
plt.rcParams['figure.figsize'] = (20, 6)
plt.style.use('fivethirtyeight')

# plotting a pie chart to represent share of Shade column
plt.subplot(1, 1, 1)
labels = set(list(df['Status']))
sizes = df['Status'].value_counts()
colors = plt.cm.Wistia(np.linspace(0, 1, 5))
explode = [0, 0, 0, 0]

plt.pie(sizes, labels = labels, colors = colors, shadow = True, startangle = 90)
plt.title('Status', fontsize = 20)


plt.legend()
plt.show()
```



We can see that "Status" column is imbalanced, we might wanna consider that in order to avoid overfitting. For now, we will not change the distribution of that column.

In [77]:
```
# check the distribution of Country

plt.rcParams['figure.figsize'] = (30, 4)
sns.countplot(x=df['Country'], palette = 'spring',order = df['Country'].value_counts().index)
plt.xlabel(' ', fontsize = 50)
plt.title('Distribution of Colors')
plt.show()
```



In [78]:
```
# Distribution of Year↔
```

We can see that the feature of "Year" is almost distributed uniformly, which can benefit to our regression model.

In [79]:
```
# for i in list(df.columns):
#     sns.histplot(data=df, x=i, kde=True, bins=16)
#     plt.show()
```

Bivariate Analysis & Multivariate Analysis

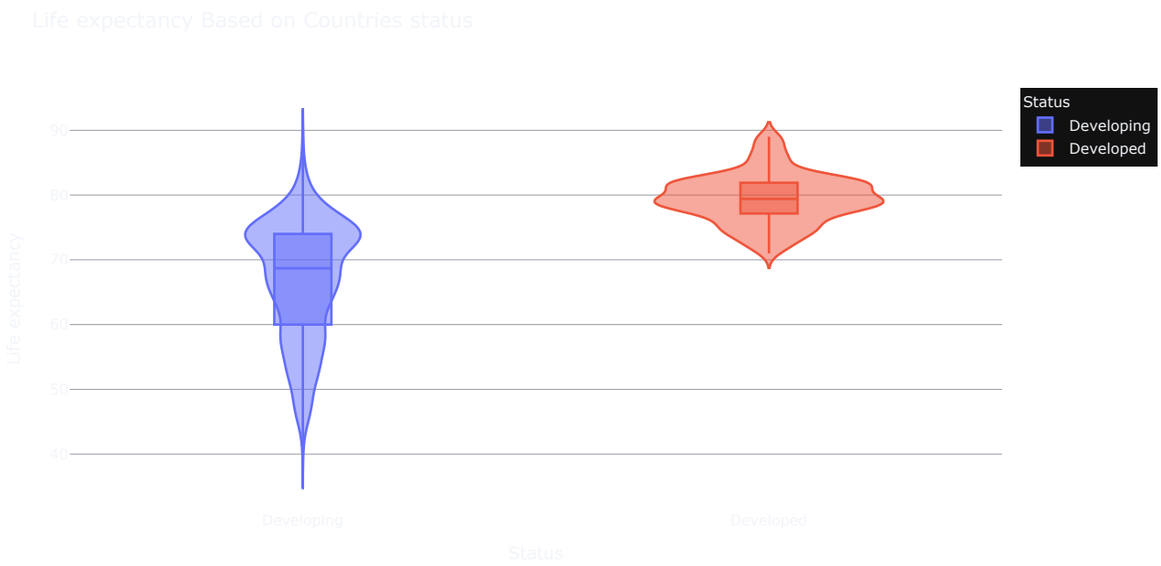
Bivariate analysis is one of the simplest forms of quantitative analysis. It involves the analysis of two variables, for the purpose of determining the empirical relationship between them. Bivariate analysis can be helpful in testing simple hypotheses of association.

- Types of Bivariate Analysis
  - Categorical vs Categorical
  - Categorical vs Numerical
  - Numerical vs Numerical
- First, we will perform Categorical vs Categorical Analysis using Stacked and Grouped Bar Charts with the help of crosstab function.
- Second, we will perform Categorical vs Numerical Analysis using Bar Charts, Box plots, Strip plots, Swarm plots, Boxen plots, Violin Plots, etc
- Atlast, we will perform Numerical vs Numerical Analysis using Scatter plots.

Multivariate analysis is based on the principles of multivariate statistics, which involves observation and analysis of more than one statistical outcome variable at a time.

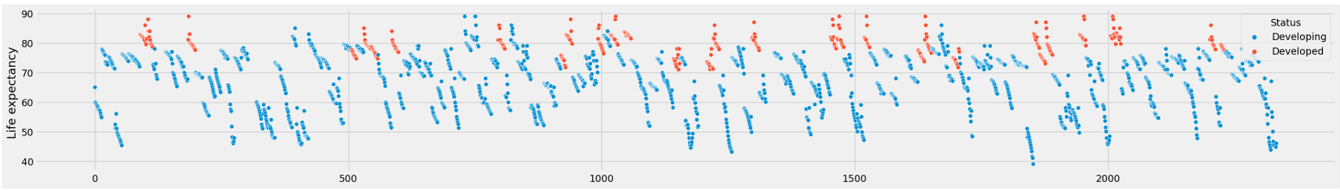
- We will use the Correlation Heatmap to check the correlation between the Numerical Columns
- We will check the ppscore or the Predictive Score to check the correlation between all the columns present in the data.
- We will use Bubble Charts, split Violin plots, Hue with Bivariate Plots.

In [80]:
```
px.violin(df,x='Status',y='Life expectancy ',color='Status',template='plotly_dark',
          box=True,title='Life expectancy Based on Countries status')
```



In [81]:
```
sns.scatterplot(data=df, x=df.index, y="Life expectancy ", hue="Status")
```

Out[81]:
```
<AxesSubplot:ylabel='Life expectancy '>
```



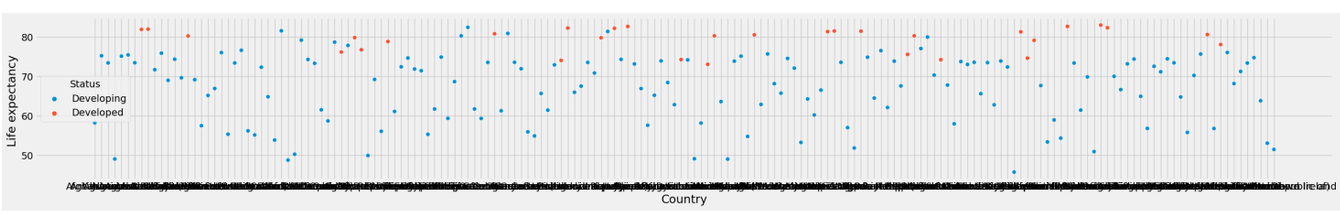
In [82]:
```
df.groupby('Country').agg({
    'Status':'first',
    'Life expectancy ':'mean',
})
```

Out[82]:

Country	Status	Life expectancy
Afghanistan	Developing	58.178571
Albania	Developing	75.208333
Algeria	Developing	73.400000
Angola	Developing	48.985714
Antigua and Barbuda	Developing	75.115385
Argentina	Developing	75.445455
Armenia	Developing	73.436364
Australia	Developed	81.915385
Austria	Developed	81.984615
Azerbaijan	Developing	71.677778

In [83]:
```
sns.scatterplot(data=df.groupby('Country').agg({
    'Status':'first',
    'Life expectancy ':'mean',
}), x=df.groupby('Country').agg({
    'Status':'first',
    'Life expectancy ':'mean',
}).index, y="Life expectancy ", hue="Status")
```

Out[83]:
```
<AxesSubplot:xlabel='Country', ylabel='Life expectancy '>
```



It is clearly that Developed countries has an higher Life expectancy

In [84]:
```
# px.scatter(df,y='Adult Mortality',x='Life expectancy ',color='Country',size='Life expectancy ',template='plotly_dark',opacity=0.6,title='<b>
```

In [85]:
```
df[df['Adult Mortality'] > 100][['Adult Mortality','Life expectancy ']].corr()
```

Out[85]:

	Adult Mortality	Life expectancy
Adult Mortality	1.000000	-0.912517
Life expectancy	-0.912517	1.000000

We can see that regardless the outliers, meaning without the observation that their Adult Mortality is < 100, we get that Adult Mortality is highly correlated with out target variable - Life exxpectancy.

Dealing with Categorical Columns - OneHot Encoding & Label Encoding

Categorical variables are known to hide and mask lots of interesting information in a data set. It's crucial to learn the methods of dealing with such variables. If you won't, many a times, you'd miss out on finding the most important variables in a model. It has happened with me. Initially, I used to focus more on numerical variables. Hence, never actually got an accurate model. But, later I discovered my flaws and learnt the art of dealing with such variables.

- There are various ways to encode categorical columns into Numerical columns
- This is an Essential Step, as we Machine Learning Models only works with Numerical Values.
- Here, we are going to use Business Logic to encode the education column
- Then we will use the Label Encoder, to Department and Gender Columns

In [86]:
```
# check the categorical columns present in the data
df.select_dtypes('object').columns
```

Out[86]:
```
Index(['Country', 'Status'], dtype='object')
```

In [87]:
```
# check the value counts for the Country column
df['Country'].value_counts()
```

Out[87]:
```
Afghanistan                         16
Sao Tome and Principe               16
Niger                               16
Nigeria                             16
Norway                              16
Oman                                16
Pakistan                            16
Panama                              16
Papua New Guinea                    16
Peru                                16
Philippines                         16
Poland                              16
Portugal                            16
Qatar                               16
Republic of Korea                   16
Republic of Moldova                 16
Romania                             16
Russian Federation                  16
Rwanda                              16
Saint Lucia                         16
Saint Vincent and the Grenadines    16
Nicaragua                           16
New Zealand                         16
Netherlands                         16
Malta                               16
Liberia                             16
Lithuania                           16
Luxembourg                          16
Madagascar                          16
Malawi                              16
                                    ..
Japan                               16
Jordan                              16
Kazakhstan                          16
Kenya                               16
Kiribati                            16
Kuwait                              16
Kyrgyzstan                          16
Haiti                               16
Guinea-Bissau                       16
Dominican Republic                  16
Finland                             16
Ecuador                             16
Egypt                               16
El Salvador                         16
Equatorial Guinea                   16
Eritrea                             16
Estonia                             16
Ethiopia                            16
Fiji                                16
France                              16
Guinea                              16
Gabon                               16
Gambia                              16
Georgia                             16
Germany                             16
Ghana                               16
Greece                              16
Grenada                             16
Guatemala                           16
Zimbabwe                            13
Name: Country, Length: 178, dtype: int64
```

In [88]:
```
df.isnull().sum()
```

Out[88]:
```
Country                             0
Year                                0
Status                              0
Life expectancy                   595
Adult Mortality                     0
infant deaths                       0
percentage expenditure              0
Hepatitis B                         0
 BMI                                0
under-five deaths                   0
Polio                               0
Diphtheria                          0
 HIV/AIDS                           0
GDP                                 0
 thinness  1-19 years               0
 thinness 5-9 years                 0
Income composition of resources     0
Schooling                           0
ID                               2250
dtype: int64
```

In [89]:
```
# encoding these categorical columns to convert them into numerical columns

dummies1 = pd.get_dummies(df['Country'])
df = pd.concat([df, dummies1], axis=1)

df = df.drop(['Country'], axis = 1)

# dummies2 = pd.get_dummies(df['Status'])
# df = pd.concat([df, dummies2], axis=1)

# df = df.drop(['Status'], axis = 1)
```

In [90]:
```
# We Can also use Label Encoding for Country to convert them into Numerical,
# but we wan't to have a dicionary for ourselves so we can know what each number represent.

# use Label Encoding for Direction to convert them into Numerical,
le = LabelEncoder()
df['Status'] = le.fit_transform(df['Status'])

# df['Country'] = le.fit_transform(df['Country'])

# test['Status'] = le.fit_transform(test['Status'])

# Check the data after encoding

# df = df.drop(['Country'], axis = 1)
# test = test.drop(['Country'], axis = 1)
```

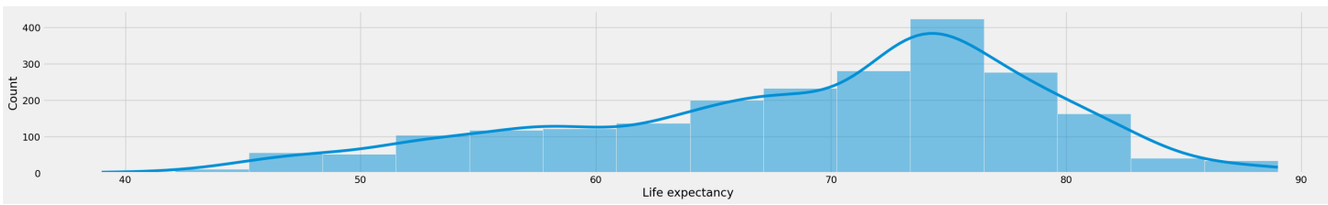
Splitting the Data

This is one of the most Important step to perform Machine Learning Prediction on a Dataset, We have to separate the Target and Independent Columns.

- We store the Target Variable in y, and then we store athe rest of the columns in x, by deleting the target column from the data
- Also, we are changing the name of test dataset to x_test for ease of understanding.

In [91]:
```
sns.histplot(data=df, x="Life expectancy ", kde=True, bins=16)
```

Out[91]:
```
<AxesSubplot:xlabel='Life expectancy ', ylabel='Count'>
```



In [92]:
```
df1 = df[df['ID'].isna()]

test = df[df['ID'].notnull()]

df = df1.drop(['ID'], axis = 1)
```

In [93]:
```
# Split the target data from the df

y = df['Life expectancy ']
x = df.drop(['Life expectancy '], axis = 1)

# Print the shapes of these newly formed data sets
print("Shape of the x :", x.shape)
print("Shape of the y :", y.shape)

# Create a validation set from the training data so that we can check whether the model that we have created is good enough

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.001, random_state = 42)

# Print the shapes
print("Shape of the x Train :", x_train.shape)
print("Shape of the y Train :", y_train.shape)
print("Shape of the x Valid :", x_test.shape)
print("Shape of the y Valid :", y_test.shape)
```

```
Shape of the x : (2250, 194)
Shape of the y : (2250,)
Shape of the x Train : (2247, 194)
Shape of the y Train : (2247,)
Shape of the x Valid : (3, 194)
Shape of the y Valid : (3,)
```

In [94]:
```
print("Dimension of Train set as of now is ",x_train.shape)
```

```
Dimension of Train set as of now is  (2247, 194)
```

In [95]:
```
# X_train = preprocessing.scale(x_train)
# X_test = preprocessing.scale(x_test)

sc = StandardScaler()
# X_train = sc.fit_transform(x_train)
# X_test = sc.transform(x_test)
X_train = x_train
X_test = x_test

# test = preprocessing.scale(test)
```

In [96]:
```
test = test.set_index('ID')

test = test.drop(['Life expectancy '], axis = 1)
```

Linear Regression

In [97]:
```
lr = LinearRegression()

linear_reg_model = lr.fit(X_train,y_train)

y_train_pred = linear_reg_model.predict(X_train)
y_test_pred = linear_reg_model.predict(X_test)

print("The training R^2 score for linear model ",r2_score(y_train, y_train_pred))
print("The training RMSE score for linear model ",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for linear model  0.9671988036194281
The training RMSE score for linear model  1.7418112119478404
```

In [98]:
```
n_components = list(range(1,X_train.shape[1]+1,1))
pca = decomposition.PCA()

steps = [
#     ('scalar', StandardScaler()),
#     ("pca", pca),
#     ('poly', PolynomialFeatures(degree=2)),
    ('model', LinearRegression())
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)
# Predicting the Test set results
y_pred = ridge_pipe.predict(X_test)

accuracies = cross_val_score(estimator = ridge_pipe, X = X_train, y = y_train, cv = 20)

parameters = [ {
#     'pca__n_components': [1,2, 3, 4,5,6,7,8],
    'model__fit_intercept': [True, False],
    'model__normalize': [True, False]
} ]

scoring_func = make_scorer(mean_squared_error)
grid_search = HalvingGridSearchCV(estimator = ridge_pipe,
                          param_grid = parameters,
                          scoring = scoring_func,
                          n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)  # <-- GETTING ERROR IN HERE

y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

print("The training R^2 score for best Optimized Ridge model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Optimized Ridge model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Optimized Ridge model is 0.9671987934909302

The training RMSE score for best Optimized Ridge model is 1.74181148086989
```

In [99]:
```
pd.DataFrame(lr.predict(test)).to_csv('Output.csv')
```

Ridge

In [100]:
```
ridge = Ridge()

param_grid = {
            "alpha": [1e0,0.3,0.5, 0.1, 1e-2, 1e-3,1e-4,0.9,0.2,0.05],
            'tol': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
            'normalize' : [True, False],
            'solver' : ["auto", "svd", "cholesky", "lsqr", "sparse_cg", "sag", "saga"]
             }
grid = HalvingGridSearchCV(ridge, param_grid, refit = True, verbose = -1, n_jobs=-1, return_train_score = True)

ridge_best_model = grid.fit(X_train, y_train)

print(grid.best_params_)
```

```
max_resources_: 2247
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 2100
n_resources: 10
----------
iter: 1
n_candidates: 700
n_resources: 30
----------
iter: 2
n_candidates: 234
n_resources: 90
----------
iter: 3
n_candidates: 78
n_resources: 270
```

In [101]:
```
y_train_pred = ridge_best_model.predict(X_train)
y_test_pred = ridge_best_model.predict(X_test)

print("The training R^2 score for best Ridge Regression model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Ridge Regression model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Ridge Regression model is 0.9404264912456416

The training RMSE score for best Ridge Regression model is 2.3473769165467506
```

In [102]:
```
n_components = list(range(1,X_train.shape[1]+1,1))
pca = decomposition.PCA()

steps = [
#     ('scalar', StandardScaler()),
#     ("pca", pca),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Ridge())
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)
# Predicting the Test set results
y_pred = ridge_pipe.predict(X_test)

accuracies = cross_val_score(estimator = ridge_pipe, X = X_train, y = y_train, cv = 10)

parameters = [ {'model__alpha': [0.9,0.5,0.3, 0.1, 0.001, 0.0001],#np.arange(0, 0.9, 0.1),
#                 'pca__n_components': [1,2, 3, 4,5,6,7,8],
                           'model__tol': [1,3,5,7,9,10,12,14,15],
             'model__normalize' : [True, False],
             'model__solver' : ["auto", "svd", "cholesky"]}]# "lsqr", "sparse_cg", "sag", "saga"]} ]

scoring_func = make_scorer(mean_squared_error)
grid_search = HalvingGridSearchCV(estimator = ridge_pipe,
                           param_grid = parameters,
                           scoring = scoring_func,
                           cv = 10,
                           n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)  # <-- GETTING ERROR IN HERE
y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

print("The training R^2 score for best Optimized Ridge model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Optimized Ridge model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Optimized Ridge model is 0.970145360149056

The training RMSE score for best Optimized Ridge model is 1.6617364946141595
```

In [103]:
```
pd.DataFrame(grid_search.predict(test)).to_csv('Output.csv')
```

Kernel Ridge

In [104]:
```
from sklearn.kernel_ridge import KernelRidge

krr = KernelRidge()

param_grid = {
            "kernel":['linear','rbf'],
            "alpha": [1, 0.3,0.1, 1e-2, 1e-3,1e-4,1e-5],
            "gamma": np.logspace(-2, 2, 8),
            "degree": [0, 1,2,3,4,5,6,7,8, 9,10]
             }
grid = HalvingGridSearchCV(krr, param_grid, refit = True, verbose = -1, n_jobs=-1, return_train_score = True)

krr_best_model = grid.fit(X_train, y_train)

print(grid.best_params_)
```

```
max_resources_: 2247
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 1232
n_resources: 10
----------
iter: 1
n_candidates: 411
n_resources: 30
----------
iter: 2
n_candidates: 137
n_resources: 90
----------
iter: 3
n_candidates: 46
n_resources: 270
```

In [105]:
```
y_train_pred = krr_best_model.predict(X_train)
y_test_pred = krr_best_model.predict(X_test)

print("The training R^2 score for best Kernel Ridge model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Kernel Ridge model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Kernel Ridge model is 0.9559584720896008

The training RMSE score for best Kernel Ridge model is 2.0183079378372364
```

In [106]:
```
n_components = list(range(1,X_train.shape[1]+1,1))
pca = decomposition.PCA()

steps = [
    ('scalar', StandardScaler()),
    ("pca", pca),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', KernelRidge())
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)
# Predicting the Test set results
y_pred = ridge_pipe.predict(X_test)

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = ridge_pipe, X = X_train, y = y_train, cv = 10)

parameters = [ {'model__alpha': np.arange(0, 0.9, 0.1),
                'pca__n_components': [1,2, 3, 4,5,6,7,8],
                           "model__kernel":['linear'],
            "model__gamma": np.logspace(-2, 2, 5),
            "model__degree": [4]} ]

from sklearn.metrics import fbeta_score, make_scorer
scoring_func = make_scorer(mean_squared_error)
grid_search = HalvingGridSearchCV(estimator = ridge_pipe,
                          param_grid = parameters,
                          scoring = scoring_func,
                          cv = 10,
                          n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)  # <-- GETTING ERROR IN HERE
```

In [107]:
```
y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

print("The training R^2 score for best Optimized Kernel Ridge model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Optimized Kernel Ridge model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Optimized Kernel Ridge model is 0.8830660830304891

The training RMSE score for best Optimized Kernel Ridge model is 3.288716887400155
```

Lasso

In [108]:
```
lasso = Lasso()

param_grid = {
            "alpha": [0.3,0.7,0.5,1, 0.1, 1e-2, 1e-3,1e-4,1e-5],
            'tol': [0,1,2,3,4,5,6,7,8,9],
       'normalize' : [True, False],
    'selection' : ["cyclic", "random"]
              }
grid = HalvingGridSearchCV(lasso, param_grid, refit = True, verbose = -1, n_jobs=-1, return_train_score = True)

lasso_best_model = grid.fit(X_train, y_train)

print(grid.best_params_)
```

```
n_iterations: 5
n_required_iterations: 6
n_possible_iterations: 5
min_resources_: 10
max_resources_: 2247
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 360
n_resources: 10
----------
iter: 1
n_candidates: 120
n_resources: 30
----------
iter: 2
n_candidates: 40
n_resources: 90
----------
iter: 3
n_candidates: 14
n_resources: 270
----------
iter: 4
n_candidates: 5
n_resources: 810
{'alpha': 0.5, 'normalize': False, 'selection': 'cyclic', 'tol': 1}
```

In [109]:
```
y_train_pred = lasso_best_model.predict(X_train)
y_test_pred = lasso_best_model.predict(X_test)

print("The training R^2 score for best Lasso Regression model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Lasso Regression model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Lasso Regression model is 0.8032085835679175

The training RMSE score for best Lasso Regression model is 4.266377844888721
```

In [110]:
```
n_components = list(range(1,X_train.shape[1]+1,1))
pca = decomposition.PCA()

steps = [
    ('scalar', StandardScaler()),
    ("pca", pca),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso())
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)
# Predicting the Test set results
y_pred = ridge_pipe.predict(X_test)

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = ridge_pipe, X = X_train, y = y_train, cv = 10)

parameters = [ {'model__alpha': np.arange(0, 0.1, 0.03),
                'model__tol': [0,1,2,3,4,5,6,7,8,9,10],
        'model__normalize' : [True, False],
    'model__selection' : ["cyclic", "random"]} ]

from sklearn.metrics import fbeta_score, make_scorer
scoring_func = make_scorer(mean_squared_error)
grid_search = HalvingGridSearchCV(estimator = ridge_pipe,
                          param_grid = parameters,
                          scoring = scoring_func,
                          cv = 10,
                          n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)  # <-- GETTING ERROR IN HERE
```

In [111]:
```
y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

print("The training R^2 score for best Optimized Lasoo model is",r2_score(y_train, y_train_pred))
print()
print("The training RMSE score for best Optimized Lasso model is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for best Optimized Lasoo model is 0.9390450473019877

The training RMSE score for best Optimized Lasso model is 2.374437479924095
```

In [112]:
```
pd.DataFrame(lasso_best_model.predict(test)).to_csv('Output.csv')
```

In [113]:
```
for alpha in range(1,3):

    polynomial_regressor = PolynomialFeatures(degree = alpha)

    X_poly_train = polynomial_regressor.fit_transform(X_train)
    X_poly_test = polynomial_regressor.fit_transform(X_test)

    poly_reg_model = Ridge().fit(X_poly_train,y_train)

    y_train_pred = poly_reg_model.predict(X_poly_train)
    y_test_pred = poly_reg_model.predict(X_poly_test)

    train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
    test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

    print("The training R^2 score for polynomial model with degree ",alpha, "is",r2_score(y_train, y_train_pred))
    print("The test R^2 score for polynomial model with degree ",alpha, "is",r2_score(y_test, y_test_pred))
    print("The training RMSE score for polynomial model with degree ",alpha, "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
    print("The test RMSE score for polynomial model with degree ",alpha, "is",math.sqrt(mean_squared_error(y_test_pred,y_test)))
    print()
```

```
The training R^2 score for polynomial model with degree  1 is 0.9593970796433032
The test R^2 score for polynomial model with degree  1 is 0.8088714978076108
The training RMSE score for polynomial model with degree  1 is 1.9379156734461862
The test RMSE score for polynomial model with degree  1 is 0.5201471071758931

The training R^2 score for polynomial model with degree  2 is 0.9691476223340301
The test R^2 score for polynomial model with degree  2 is 0.5502595905022448
The training RMSE score for polynomial model with degree  2 is 1.6892757942499628
The test RMSE score for polynomial model with degree  2 is 0.7978925586959569
```

In [114]:
```
for alpha in range(1,3):

    polynomial_regressor = PolynomialFeatures(degree = alpha)

    X_poly_train = polynomial_regressor.fit_transform(X_train)
    X_poly_test = polynomial_regressor.fit_transform(X_test)

    poly_reg_model = Lasso().fit(X_poly_train,y_train)

    y_train_pred = poly_reg_model.predict(X_poly_train)
    y_test_pred = poly_reg_model.predict(X_poly_test)

    train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
    test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

    print("The training R^2 score for polynomial model with degree ",alpha, "is",r2_score(y_train, y_train_pred))
    print("The test R^2 score for polynomial model with degree ",alpha, "is",r2_score(y_test, y_test_pred))
    print("The training RMSE score for polynomial model with degree ",alpha, "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
    print("The test RMSE score for polynomial model with degree ",alpha, "is",math.sqrt(mean_squared_error(y_test_pred,y_test)))
    print()
```

```
The training R^2 score for polynomial model with degree  1 is 0.8219395391583751
The test R^2 score for polynomial model with degree  1 is -0.3790607354718565
The training RMSE score for polynomial model with degree  1 is 4.058261108658312
The test RMSE score for polynomial model with degree  1 is 1.3971889942114941

The training R^2 score for polynomial model with degree  2 is 0.9747970396706304
The test R^2 score for polynomial model with degree  2 is 0.7007091196978397
The training RMSE score for polynomial model with degree  2 is 1.5267995297269095
The test RMSE score for polynomial model with degree  2 is 0.650893899448165
```

In [115]:
```
polynomial_regressor = PolynomialFeatures(degree = 2)

X_poly_train = polynomial_regressor.fit_transform(X_train)
X_poly_test = polynomial_regressor.fit_transform(X_test)

param_grid = {
            "alpha": [0.0001,0.001,0.1,0.5,0.9],
            'tol': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
            'normalize' : [True, False],
            'solver' : ["auto", "svd", "cholesky"]#, "lsqr", "sparse_cg", "sag", "saga"]
             }

grid = HalvingGridSearchCV(Ridge(), param_grid, refit = True, verbose = -1, n_jobs=-1, return_train_score = True)

best_model = grid.fit(X_poly_train,y_train)

print(grid.best_params_)

y_train_pred = poly_reg_model.predict(X_poly_train)
y_test_pred = poly_reg_model.predict(X_poly_test)

train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

print("The training R^2 score for polynomial model with degree 2", "is",r2_score(y_train, y_train_pred))
print("The training RMSE score for polynomial model with degree 2", "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
n_iterations: 5
n_required_iterations: 6
n_possible_iterations: 5
min_resources_: 10
max_resources_: 2247
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 450
n_resources: 10
----------
iter: 1
n_candidates: 150
n_resources: 30
----------
iter: 2
n_candidates: 50
n_resources: 90
----------
iter: 3
n_candidates: 17
n_resources: 270
----------
iter: 4
n_candidates: 6
n_resources: 810
{'alpha': 0.1, 'normalize': True, 'solver': 'svd', 'tol': 11}
The training R^2 score for polynomial model with degree 2 is 0.9747970396706304
The training RMSE score for polynomial model with degree 2 is 1.5267995297269095
```

In [116]:
```
polynomial_regressor = PolynomialFeatures(degree = 2)

X_poly_train = polynomial_regressor.fit_transform(X_train)
X_poly_test = polynomial_regressor.fit_transform(X_test)

poly_reg_model = KernelRidge(
    alpha=0.0001,
    kernel='linear',
    gamma=100,
    degree=100,
    coef0=100,
).fit(X_poly_train,y_train)

y_train_pred = poly_reg_model.predict(X_poly_train)
y_test_pred = poly_reg_model.predict(X_poly_test)

train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

print("The training R^2 score for polynomial model with degree 2", "is",r2_score(y_train, y_train_pred))
print("The training RMSE score for polynomial model with degree 2", "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for polynomial model with degree 2 is 0.9646844356294523
The training RMSE score for polynomial model with degree 2 is 1.8073377303164089
```

In [117]:
```
pd.DataFrame(poly_reg_model.predict(polytest)).to_csv('Output.csv')
```

In [ ]:
```
polynomial_regressor = PolynomialFeatures(degree = 2)

X_poly_train = polynomial_regressor.fit_transform(X_train)
X_poly_test = polynomial_regressor.fit_transform(X_test)

poly_reg_model = Ridge(
    alpha=0.06,
    fit_intercept=True,
    normalize='deprecated',
    copy_X=True,
    max_iter=None,
    tol=0.001,
    solver='auto',
    positive=False,
    random_state=10,
).fit(X_poly_train,y_train)

y_train_pred = poly_reg_model.predict(X_poly_train)
y_test_pred = poly_reg_model.predict(X_poly_test)

train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

print("The training R^2 score for polynomial model with degree 2", "is",r2_score(y_train, y_train_pred))
print("The training RMSE score for polynomial model with degree 2", "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

The only required model

In [119]:
```
polynomial_regressor = PolynomialFeatures(degree = 2)

X_poly_train = polynomial_regressor.fit_transform(X_train)
X_poly_test = polynomial_regressor.fit_transform(X_test)

poly_reg_model = Lasso(
    alpha=0.14,
    fit_intercept=True,
    normalize='deprecated',
    precompute=False,
    copy_X=False,
    max_iter=1600,
    tol=0.000001,
    warm_start=False,
    positive=False,
    random_state=10,
    selection='cyclic',
).fit(X_poly_train,y_train)

y_train_pred = poly_reg_model.predict(X_poly_train)
y_test_pred = poly_reg_model.predict(X_poly_test)

train_score = math.sqrt(mean_squared_error(y_train_pred,y_train))
test_score = math.sqrt(mean_squared_error(y_test_pred,y_test))

print("The training R^2 score for polynomial model with degree ", "is",r2_score(y_train, y_train_pred))
print("The training RMSE score for polynomial model with degree ", "is",math.sqrt(mean_squared_error(y_train_pred,y_train)))
```

```
The training R^2 score for polynomial model with degree  is 0.9789062810930131
The training RMSE score for polynomial model with degree  is 1.3967954663953475
```

In [120]:
```
polytest = polynomial_regressor.fit_transform(test)
```

In [121]:
```
pd.DataFrame(poly_reg_model.predict(polytest)).to_csv('Output.csv')
```

In [ ]: