In [1]:
```
# The dataset is supervised(1).csv
# The dataset includes multiple variables which show main details about different coun
# The purpose is to predict the outcome of life expectancy (the last colums) using eig
# 'Adult_Mortality', 'Infant_deaths', 'Alcohol', 'Percentage_expenditure', 'Hepatitis_
#'Income_composition_of_resources', 'Schooling',

# The following libraries will be used
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns # for visualizing using unique plots for random distributions
import datetime as dt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
#Read the file (supervised(1).csv)
df = pd.read_csv("supervised (1).csv")
df.head(5)
```

Out[2]:

	Country	Status	Adult_Mortality	Infant_deaths	Alcohol	Percentage_expenditure	Hepatitis_B
0	Cook Islands	Developing	NaN	0	0.01	0.000000	98.0
1	Dominica	Developing	NaN	0	0.01	11.419555	96.0
2	Marshall Islands	Developing	NaN	0	0.01	871.878317	8.0
3	Monaco	Developing	NaN	0	0.01	0.000000	99.0
4	Nauru	Developing	NaN	0	0.01	15.606596	87.0

5 rows × 21 columns

In [3]:
```
print(list(df.columns))
```

```
['Country', 'Status', 'Adult_Mortality', 'Infant_deaths', 'Alcohol', 'Percentage_expe
nditure', 'Hepatitis_B', 'Measles', 'BMI', 'Under_five_deaths', 'Polio', 'Total_expen
diture', 'Diphtheria', 'HIV_AIDS', 'GDP', 'Population', 'Thinness_1_19_years', 'Thinn
ess_5_9_years', 'Income_composition_of_resources', 'Schooling', 'Life_Expectancy']
```

In [8]:
```
#Understanding the data
df.shape #Data size
df.info()#more details on the data showing presence of null values and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193 entries, 0 to 192
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Country                          193 non-null    object 
 1   Status                           193 non-null    object 
 2   Adult_Mortality                  193 non-null    float64
 3   Infant_deaths                    193 non-null    int64  
 4   Alcohol                          193 non-null    float64
 5   Percentage_expenditure           193 non-null    float64
 6   Hepatitis_B                      193 non-null    float64
 7   Measles                          193 non-null    int64  
 8   BMI                              193 non-null    float64
 9   Under_five_deaths                193 non-null    int64  
 10  Polio                            193 non-null    int64  
 11  Total_expenditure                193 non-null    float64
 12  Diphtheria                       193 non-null    int64  
 13  HIV_AIDS                         193 non-null    float64
 14  GDP                              193 non-null    float64
 15  Population                       193 non-null    float64
 16  Thinness_1_19_years              193 non-null    float64
 17  Thinness_5_9_years               193 non-null    float64
 18  Income_composition_of_resources  193 non-null    float64
 19  Schooling                        193 non-null    float64
 20  Life_Expectancy                  193 non-null    float64
dtypes: float64(14), int64(5), object(2)
memory usage: 31.8+ KB
```

In [9]:
```
#cleaning data by eliminating the missing values
df.isnull().sum()
df.fillna (df.mean(), inplace=True)#Replacing null values with the mean
df.isnull().sum()#null values are now eliminated
df.head(5)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_712\1535581083.py:3: FutureWarnin
g: The default value of numeric_only in DataFrame.mean is deprecated. In a future ver
sion, it will default to False. In addition, specifying 'numeric_only=None' is deprec
ated. Select only valid columns or specify the value of numeric_only to silence this
warning.
  df.fillna (df.mean(), inplace=True)#Replacing null values with the mean
```

Out[9]:

	Country	Status	Adult_Mortality	Infant_deaths	Alcohol	Percentage_expenditure	Hepatitis_B
0	Cook Islands	Developing	152.863388	0	0.01	0.000000	98.0
1	Dominica	Developing	152.863388	0	0.01	11.419555	96.0
2	Marshall Islands	Developing	152.863388	0	0.01	871.878317	8.0
3	Monaco	Developing	152.863388	0	0.01	0.000000	99.0
4	Nauru	Developing	152.863388	0	0.01	15.606596	87.0

5 rows × 21 columns

In [10]:
```
#Extracting the target variable checking the last column

y = df['Life_Expectancy'].values # Target variable
```

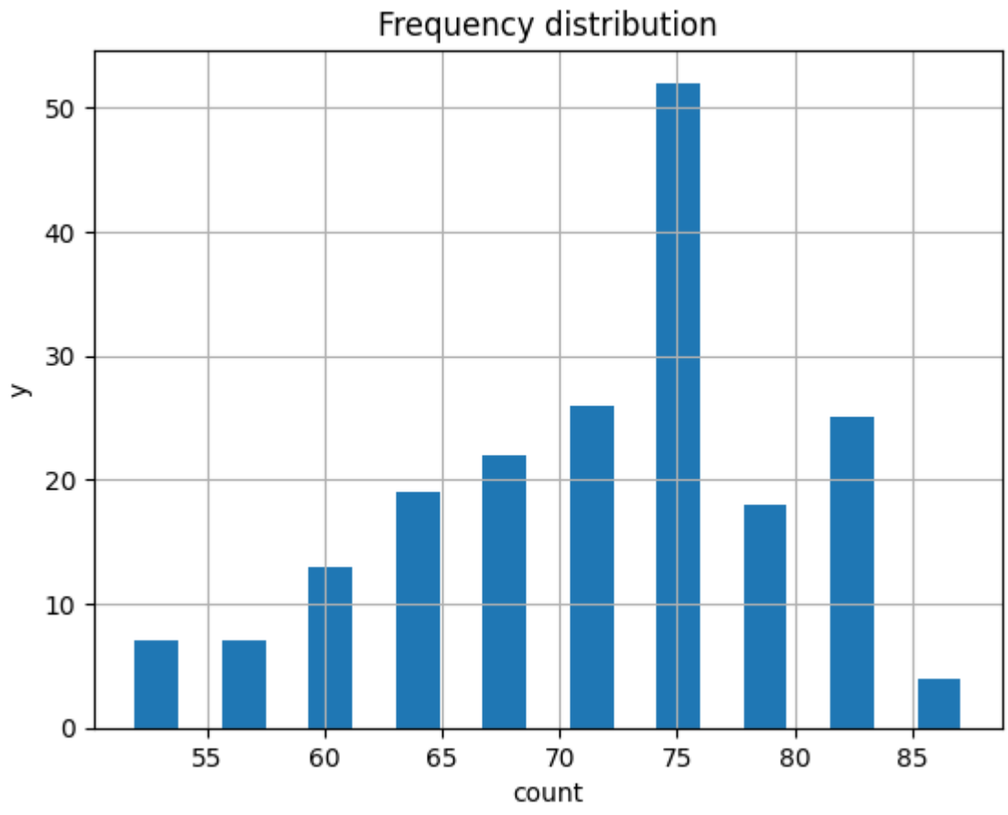
The supervised challenge is a regression task due to the numeric and continous variables as shown in the frequency distribution table.

In [11]:
```
#Generating the frequency table of y
#printing y shows that the values of the target variable is numeric.
series_of_y = pd.Series(y) # converted the list to series
y1 = pd.Series(y)
y1.describe()
```

Out[11]:
```
count    193.000000
mean      71.616940
std        7.909323
min       51.000000
25%       66.100000
50%       73.300000
75%       76.700000
max       88.000000
dtype: float64
```

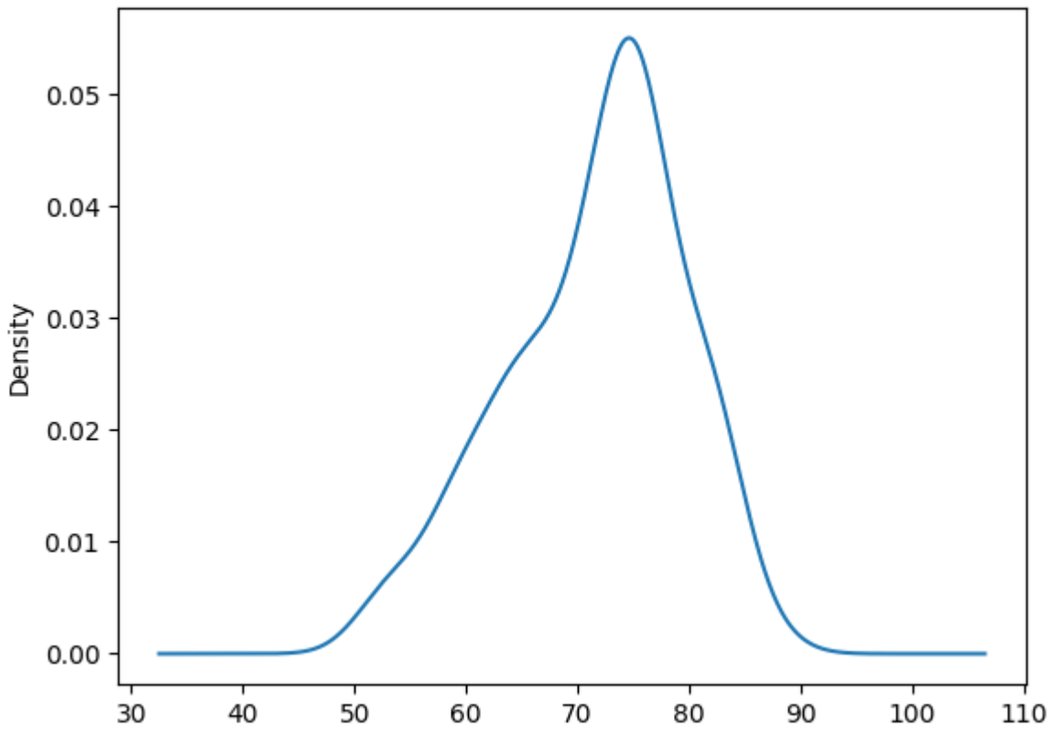
In [12]:
```
#Visualizing the frequency table
import matplotlib.pyplot as plt
y1.plot.hist( grid=True, bins=10, rwidth=0.5)
plt.title('Frequency distribution')
plt.ylabel('y')
plt.xlabel('count')
```

Out[12]:
```
Text(0.5, 0, 'count')
```



In [13]:
```
#plot the density function
y1.plot.density()
```

Out[13]:
```
<AxesSubplot: ylabel='Density'>
```



The data is continous making the distribution Normal due to the bellshape

In [33]:
```
#After removing the target, all remaining attributes are your features (independent va
df. head (5)
df.drop(['Country'], axis=1)
df1= df.drop(['Country'], axis=1)
df1.head()
df2=df1.drop(['Status'], axis=1)
df2.head()
df2.shape
df2.info()
```

Out[33]:
```
(193, 19)
```

In [15]:
```
#Discuss the shape of the feature data, how many rows and columns does it contain?
# save the values of the first 18 variables into x1

x1= df2.iloc[: , :18].values
y2= df2.iloc[: , -1].values
print(x1.shape)
```

```
(193, 18)
```

Standardizing data using the StandardScaler().fit_transform function will be essential in the next codes.

In [24]:
```
# standardizing the data (both x1 and y2) using the StandardScaler().fit_transform fun
#with sklearn.preprocessing import StandardScaler
x_Stand = StandardScaler().fit_transform(x1)
y_Stand = StandardScaler().fit_transform(y2.reshape(-1, 1))
```

Splitting the data into train and test.

In [25]:
```
#splitting the data into train and test datasets with test size = 20%
# for this use the train_test_split () function, this function will return two dataset
# xTreain, xTest, yTrain, yTest
x_train, x_test, y_train, y_test = train_test_split(x_Stand, y_Stand, test_size=0.20)
y_train = y_train.reshape(-1)
y_test = y_test.reshape(-1)
print (x_train.shape)
print(y_train.shape)
print (x_test.shape)
print( y_test.shape)
```

```
(154, 18)
(154,)
(39, 18)
(39,)
```

Trainning the supervised learning model.

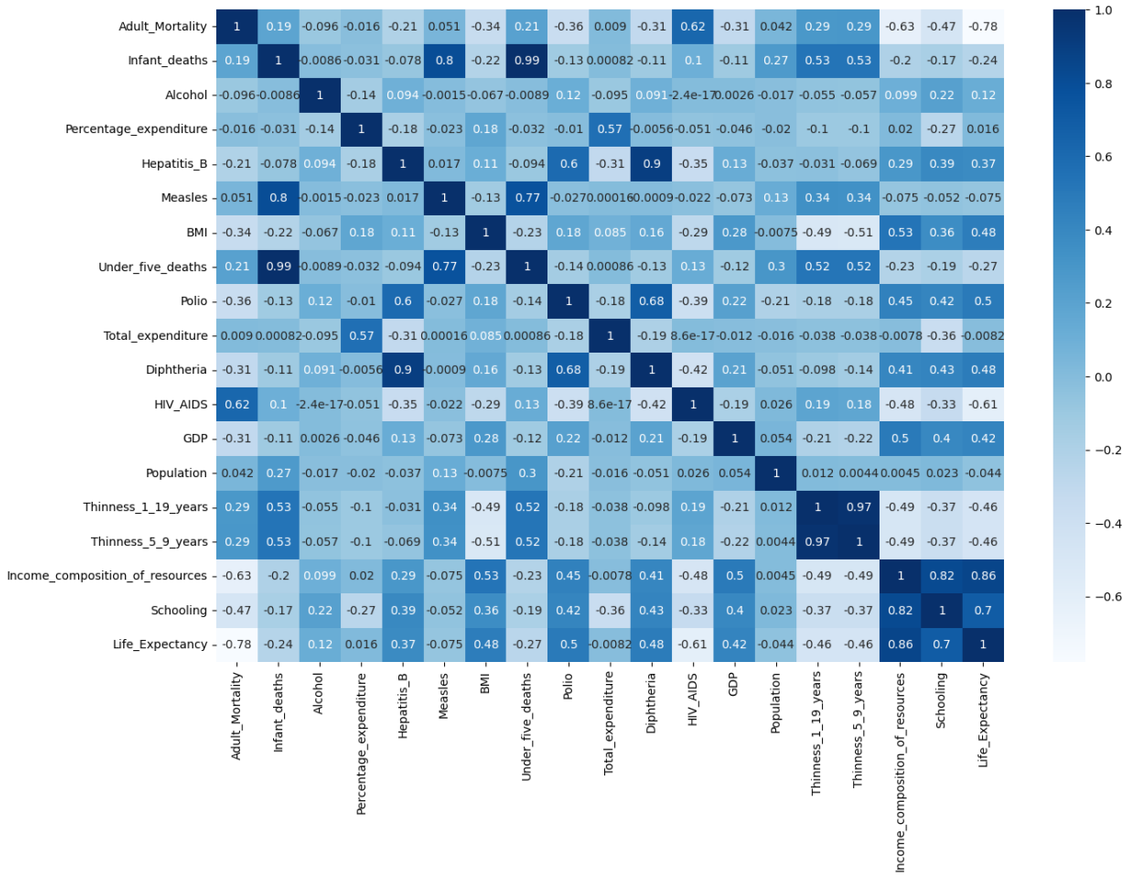
In [27]:
```
from sklearn.linear_model import LinearRegression
#apply LR model on xTrain, yTrain.
# thereafter, print the r_squared, the intercept, and the slopes using the LinearRegre
model = LinearRegression() # The OLS model
model.fit(x_train, y_train) # Using OLS on the scaled data

# print r_squared
r_sq = model.score(x_train, y_train)
print('R squared:', r_sq)


# print y-intercept (B0)
print('intercept:', model.intercept_)

#print the slopes (coef for each variable: B1,B2,B3)
print('slope:', model.coef_)

y_pred = model.predict(x_test)
print('R squared:', model.score(x_test, y_test))

# find the errors in the prediction medel
y_pred= y_pred.reshape(-1)
e = y_test - y_pred

print("MSE = ",sum(e**2)/83)
```

```
R squared: 0.8685352069734642
intercept: 0.015233303041801888
slope: [-0.31749389  0.88810181  0.03929745  0.01234989  0.08619639 -0.04069242
  0.00412995 -0.86498774  0.03180883  0.04540092  0.00444343 -0.08176462
  0.01443514  0.01104989 -0.06167839 -0.03697768  0.45726421  0.05923074]
R squared: 0.8405914854179666
MSE =  0.06356898008194546
```

In [30]:
```
#apply Ridge regression function using a range of alpha values from 0 to 100
# repeating the model for each alpha value using for loop
# in each loop, save the score for train and test, the slopes, and the intercept
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
alphas = range(0,100)
train_scores = np.array([]) # this array is to save the scores for train set when appl
test_scores = np.array([]) # this array is to save the scores for test set when applyi
coef = np.array([]) # this to save all B values corresponds to each alpha

for a in alphas:
    model_Ridge = Ridge(alpha = a)
    model_Ridge.fit(x_train, y_train)
    RidgeTrain_score = model_Ridge.score(x_train, y_train) # Get the training score
    RidgeTest_score = model_Ridge.score(x_test, y_test) # Get the test score
    train_scores = np.append(train_scores, RidgeTrain_score) # save the score for the
    test_scores = np.append(test_scores, RidgeTest_score)
    coef = np.append(coef, model_Ridge.coef_) # adding the coefficients for each varia


coef = coef.reshape(-1, 18) # reshape the array so each rwo corresponds to one alpha

tunedAlphaRidge = alphas[np.argmax(test_scores)] # return the alpha value corresponds
print("tuned Alpha Ridge on test  " , tunedAlphaRidge)

print("R2 on Ridge train: ", train_scores[tunedAlphaRidge]) # print the train score co
print("R2 on Ridge test: ",test_scores[tunedAlphaRidge]) # print the test score corres
print('variance is :', train_scores[tunedAlphaRidge]-test_scores[tunedAlphaRidge])
print("coef on best Ridge: ",coef[np.argmax(test_scores)])
```

```
tuned Alpha Ridge on test   0
R2 on Ridge train:  0.8685352069734642
R2 on Ridge test:  0.8405914854179662
variance is : 0.027943721555497958
coef on best Ridge:  [-0.31749389  0.88810181  0.03929745  0.01234989  0.08619639 -0.
04069242
  0.00412995 -0.86498774  0.03180883  0.04540092  0.00444343 -0.08176462
  0.01443514  0.01104989 -0.06167839 -0.03697768  0.45726421  0.05923074]
```

In [31]:
```
df2.corr().head() #checking for correlation
#visualization of the correlation above
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
sns.heatmap(df2.corr(),annot=True,cmap='Blues')
#Heatmap is to visualize which variables have highest impact in reducing life expectan
plt.show()
```



The heatmap shows the variables in df2 that have the highest correlation.The darker shades of blue show the combinations with the crucial variables that can be used to get a more accurate model

4. Discussion: pros/cons and time complexity of the designed pipeline

Strengths

1. Simple and comprehendable

Its intuitive and easy to interpret and implement. 2. Robust The model is relatively insensitive to outliers. Further, it can give resourceful outputs despite presence of outliers in the data 3. Efficiency The model can handle relatively large datasets. 4. Versatile It can be used for many tasks and its a flexible method for a variety of machine learning problems.

Limitations

1. Linearity assumption

The assumption on a linear relationship is not true in real world scenarios. 2. Normality assumption This is not the case in real world scenarios 3. Sensitive to outliers 4. Non-linear relationships This may lead to poor predictions 5. Multicolinearity

Time complexity for running the linear regression is O(n^3) and n is the number of observations. While implementing ridge and lasso has a time complexity that is cubic. The dataset is not very large therefore the computationaltime was not too long.For a dataset with 193 rows and 18 columns, the time complexity would be O(193^3), which would be a relatively small amount of computation. This means that OLS regression would be very fast for this size of data and could be calculated in a matter of seconds on a modern computer.

The impact of standardizing and regularizing data on time execution, in comparison to OLS regression, is contingent upon the implementation and size of the dataset. Typically, operations like scaling and transforming features in standardizing and regularizing the data have less effect on time execution. However, OLS regression can be quite time-consuming and computationally intensive, especially when dealing with large datasets. In cases where the dataset is extremely large and the OLS implementation is not optimized, standardizing and regularizing the data may also contribute to increased time execution.

Inorder to improve the model, the alternative methods would be transforming the independent variables and factor analysis which would combine the highly corelated variables(features) into a lower count of underlying factors. In PCA (Principle Component Analysis), the independent variables are converted into a new collection of uncorrelated features called principal components, which dimishes the dimensionality of our data.Orthogonal contrasts: Make the independent variables less multicollinear by how you design them. Increasing the sample size by Collecting more data can help to lessen the effects of multicollinearity in a regression model.

In [ ]: