# Machine Learning Engineer Nanodegree

Capstone Project

P.T.V.KRISHNA
JUNE 29TH, 2018

# Project Overview

- Although there have been lot of studies undertaken in the past on factors affecting life expectancy considering demographic variables, income composition and mortality rates, it was found that effect of immunization and human development index was not taken into account in the past.
- Also, some of the past research was done considering multiple linear regression based on data set of one year for all the countries. Hence, this gives motivation to resolve both the factors stated previously by formulating a regression model based on mixed effects model and multiple linear regression while considering data from a period of 2000 to 2015 for all the countries. Important immunization like Hepatitis B, Polio and Diphtheria will also be considered.
- In a nutshell, this study will focus on immunization factors, mortality factors, economic factors, social factors and other health related factors as well. Since the observations this dataset are based on different countries, it will be easier for a country to determine the predicting factor which is contributing to lower value of life expectancy. This will help in suggesting a country which area should be given importance in order to efficiently improve the life expectancy of its population.

# Problem Statement

- By accurately predicting the ‘Life expectancy’ of a country based on various factors which are correlated.
- By using the data-set on Life Expectancy, the task is to predict the accuracy of ‘Life Expectancy’ of a country based on various factors mentioned in the dataset. The model utilizes the important characteristics of the data to develop models that can predict the target variable.
- I decided to implement Machine-Learning techniques to predict the ‘Life Expectancy’ of a country by using various factors like BMI, infant and adult mortality rate, diseases, health care improvement rate etc..
- This Project constitutes Data Exploration and Visualizations, Data Preprocessing and finally testing various algorithms and techniques.

# Datasets and Inputs

- The data was collected from WHO and United Nations website with the help of Deeksha Russell and Duan Wang. It was collected from kaggle website.
- The data-set constitutes a single CSV files which is obtained from Kumarrajarshi from Kaggle
- Link :- https://www.kaggle.com/kumarajarshi/life-expectancy-who/home
- The data has 22 columns i.e., they are Country, Year, Status, BMI, Mortality rate, Hepatitis etc. and the target variable being 'Expectancy' and has 2400 rows. • We are comprised with a good amount of features which are potentially utilized to estimate good result.

# Metrics

- The current problem is a Regression task, since it takes certain features as inputs and attempts to find a score that helps an individual to get an idea about the chances of getting an admission in a specific university.
- Hence, Coefficient Of Determination is considered as the performance metric that can be applied to compare the performances of the scores obtained from the BenchMark and the Optimal Model considered.
- The CoEfficient Of Determination(R^2) is the key output of the Regression Analysis.It can be defined as the proportion of the varaince in the dependent variable that is predictable from the independent variable.
- It's values ranges from 0 to 1, and the results are given intuition by, if:-
  - The value of R^2 -> 0 , indicates that the model is a worst fit to the given data.
  - The value of R^2 -> 1 , indicates that the model is the best fit to the given data.
  - The value of R^2 in between 0 and 1 -> indicates that the respective variability exhibited by the target variable.

**IMPLEMENTATION OF THE COEFFICIENT OF DETERMINATION :**

In [205]:
```
# Import 'r2_score'
from sklearn.metrics import r2_score
def performance_metric(y_true, y_predict):
    """ Calculates and returns the performance score between
        true and predicted values based on the metric chosen. """

    # TODO: Calculate the performance score between 'y_true' and 'y_predict'
    score = r2_score(y_true,y_predict)

    # Return the score
    return score
```

I've implemented the coefficient of determination as my metric as shown in the picture above. R-squared is the "percent of variance explained" by the model. That is, **R-squared is the fraction by which the variance of the errors is less than the variance of the dependent variable.** (The latter number would be the error variance for a constant-only model, which merely predicts that every observation will equal the sample mean.) It is

called R-squared because in a simple regression model it is just *the square of the correlation* between the dependent and independent variables, which is commonly denoted by "r".

# II. Analysis

## Data Exploration

I explored the data as follows:

- First ,I imported the packages required for my project. By using pandas reasd_csv() method I loaded the whole data into a variable called 'data'.
    - Then, I used head() method to look at how the data is distributed , usually head method prints first 5 rows of the data.

- We have to load all the necessary Python libraries and also we have to get the data from csv file to a variable, probably called 'data'. Note that the 4th column from this dataset, 'Life_expectancy', will be the target label (Life expectancy of a country). All other columns are features about each country for each year in the dataset.

- Run the code cell below to load the Life expectancy dataset, along with a few of the necessary Python libraries required for this project. We will know the dataset loaded successfully if the size of the dataset is reported and display the first few entries for examination using the .head() function..

In [188]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows the use of display() for DataFrames

# Import supplementary visualization code visuals.py
# import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Census dataset
data = pd.read_csv("LifeExpectancyData.csv")

# Success - Display the first record
display(data.head(n=5))
# Success
print("Life Expectancy dataset has {} data points with {} variables each.".format(*data.shape))
```

| | Country | Year | Status | Life_expectancy | Adult Mortality | infant_deaths | Alcohol | percentage_expenditure | Hepatitis B | Measles | ... | Polio | Total_expenditure | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Afghanistan | 2015 | Developing | 65.0 | 263.0 | 62 | 0.01 | 71.279624 | 65.0 | 1154 | ... | 6.0 | 8.16 | |
| 1 | Afghanistan | 2014 | Developing | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 492 | ... | 58.0 | 8.18 | |
| 2 | Afghanistan | 2013 | Developing | 59.9 | 268.0 | 66 | 0.01 | 73.219243 | 64.0 | 430 | ... | 62.0 | 8.13 | |
| 3 | Afghanistan | 2012 | Developing | 59.5 | 272.0 | 69 | 0.01 | 78.184215 | 67.0 | 2787 | ... | 67.0 | 8.52 | |
| 4 | Afghanistan | 2011 | Developing | 59.2 | 275.0 | 71 | 0.01 | 7.097109 | 68.0 | 3013 | ... | 68.0 | 7.87 | |

5 rows × 22 columns

```
Life Expectancy dataset has 2938 data points with 22 variables each.
```

The dataset has 2938 datapoints with 22 variables each.

# Exploring Feature Set

**Country**: Name of a country.
**Year**: Year
**Status**: Developed or Developing
**Adult Mortality**: Number of deaths in adult category
**Infant-deaths**: Number of infant deaths
**Alcohol**: Alcohol consumption rate
**Percentage expenditure**: Expenditure percentage
**Hepatitis B and measles**: Number of deaths due to hepatitis and measles.
**BMI**: Average Body mass index
**Under five deaths**: Number of deaths under 5 years of age
**Polio**: Polio effected number
**Total_expenditure**: Total expenditure
**Thinness 1-19 years**: Thinness of people aged 1- 19 years.
**Thinness 5-9 years**: Thinness of people aged 5- 9 years.
**GDP** GDP of a country
**Population**: Population of a country

# Exploratory Visualization

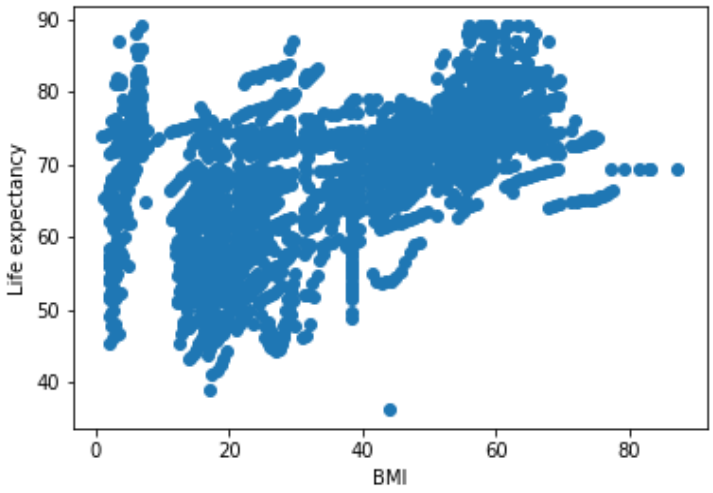
Exploratory Visualization can be defined as an approach for analyzing data sets to summarize their important characteristics, often by the application of visual methods.

The Primary theme of Exploratory Visualization is for observing what the data can give us an intuition far beyond the conventional modeling or hypothesis testing tasks.

The Following session of Data Visualization will provide the intuition of how each feature is 'Correlated' with the target varaible 'Life_expectation'
Below are the visualizations made with matplotlib.pyplot module. Each plot has correlation of a feature with respective to our target variable ‘life_expectancy’.

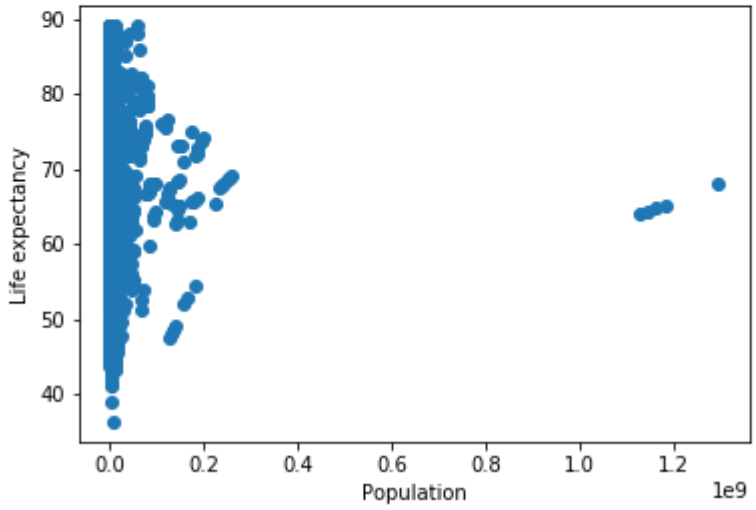
**Correlation of 'BMI' with respect to 'Life_expectancy'**

In [193]:
```
#
import matplotlib.pyplot as plt
plt.scatter(features['BMI'],life_exp)
plt.ylabel('Life expectancy')
plt.xlabel('BMI')
plt.show()
```



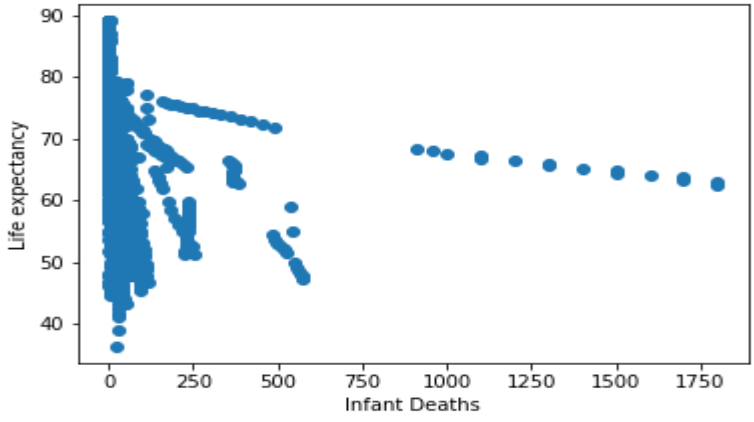
**Correlation of 'Population' with respect to 'Life_expectancy'**

In [196]:
```
#
import matplotlib.pyplot as plt
plt.scatter(features['Population'],life_exp)
plt.ylabel('Life expectancy')
plt.xlabel('Population')
plt.show()
```



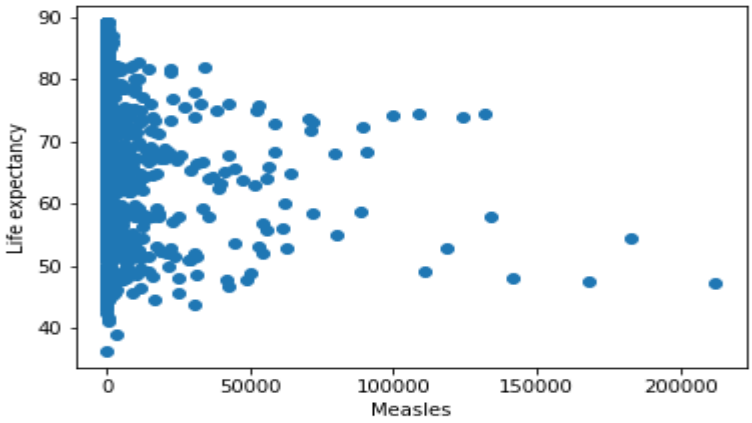

**Correlation of 'infant_deaths' with respect to 'Life_expectancy'**

In [197]:
```
import matplotlib.pyplot as plt
plt.scatter(features['infant_deaths'],life_exp)
plt.ylabel('Life expectancy')
plt.xlabel('Infant Deaths')
plt.show()
```



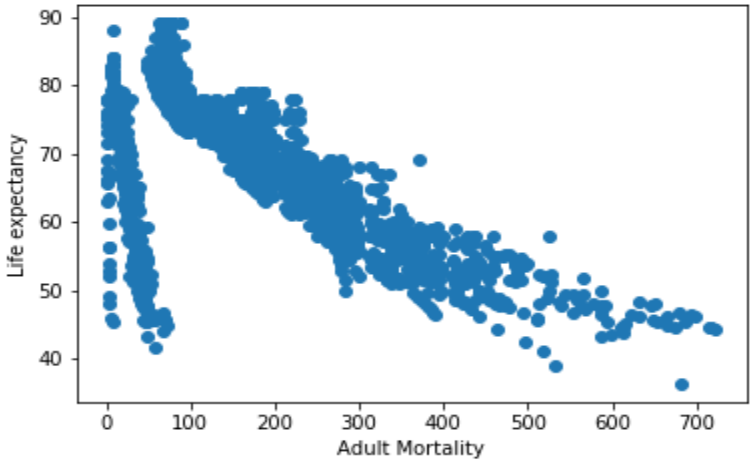
**Correlation of 'Measles' with respect to 'Life_expectancy'**

In [195]:
```
#
import matplotlib.pyplot as plt
plt.scatter(features['Measles'],life_exp)
plt.ylabel('Life expectancy')
plt.xlabel('Measles')
plt.show()
```

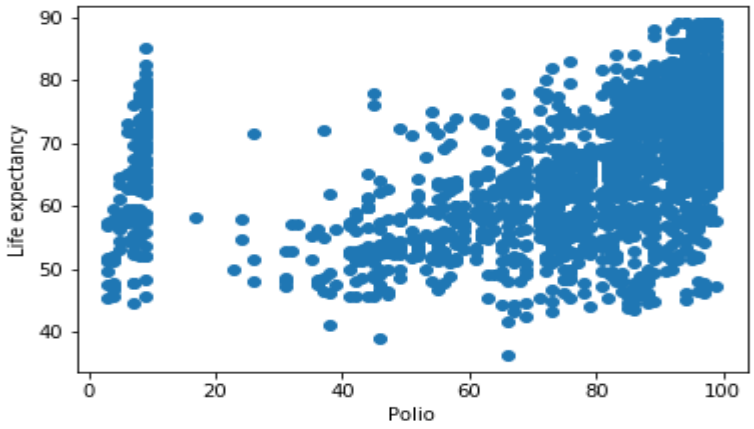



**Correlation of 'Adult Mortality' with respect to 'Life_expectancy'**

In [194]:
```
#
import matplotlib.pyplot as plt
plt.scatter(features['Adult Mortality'],life_exp)
plt.ylabel('Life expectancy')
plt.xlabel('Adult Mortality')
plt.show()
```



```
In [198]: #correlation of polio with life expectancy
          plt.scatter(features['Polio'],life_exp)
          plt.ylabel('Life expectancy')
          plt.xlabel('Polio')
          plt.show()
```




# Algorithms and Techniques

By observing the problem, it is quite evident that it is a 'Regression' Problem.
It is important to understand the intuition behind the consideration of specific model, since it has to generate an optimal possibility of results that can improve the model's performance on a sample of new data.
Hence, taking the performance of the model into consideration, I chose three Supervised Machine Learning Algorithms that can be better compatible for the data being available.
They are **:-**

- Decision Trees
- Ensemble Methods - Random Forests
- Support Vector Machines (SVM)

**Decision Trees:-**

- Decision Trees are very flexible,easy to understand and easy to debug.They usually work on Classification and Regression Problems i.e.,for categorical problems having [green,red,blue..etc.] and continuous inputs like [2.9,3.8..etc].They usually cover both the perspectives.
- It divides the dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

- As the given data is comprised of continuous features,Decision Trees perform well in regression tasks.

**Ensemble Methods - Random Forests :-**

- Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes in case of the classification tasks or mean prediction for the regression tasks of the individual trees.
- The Random forests have a stroke of brilliance when a performance optimization happens to enhance precision of the model, or vice versa.Tuning down the fraction of features that is considered at any given node can let you easily work on datasets with thousands of features.
- Since Random Forests perform well on almost every machine learning problem and they also show less overfit behavior when compared to Decision Trees. Since our problem is composed of a lot of continuous features for which Random Forests serve a better choice.

**Support Vector Machines :-**

- SVM's are simple,accurate and perform well on smaller and cleaner datasets.It can be more efficient as it uses subset of training points.
- The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. Initially as the output is a real number and continuous it becomes very difficult to predict the information at hand, which has infinite possibilities.
- In the case of regression, the factor margin of tolerance (epsilon) is set in approximation to the SVM.
- The main theme for SVM is always to minimize error, particularize the hyperplane which maximizes the margin, keeping in mind that part of the error is convinced.
- Since For the current regression problem consists of a lot of continuous features the application of SVM's can serve a better purpose.

**Data Preprocessing :-**

- It is one of the key phases in the Machine Learning Problems and the complete process and implementation steps are discussed above in the 'DATA PREPROCESSING' Phase of the project.

**Model Tuning :-**

- In this part of the project, I'll apply the GridSearchCV technique to further optimize the best model that was selected from the three supervised learning models stated above.

**Grid Search CV:-**

- Grid Search technique is an approach for tuning the parameters that are used to build and evaluate a model based on the individual combination of parameters of an algorithm specified in a grid.
- Grid Search rigorously checks for the combination of hyper-parameters in order to find the best model.
- Finally it's easy to figure out the combination that has high cross validation accuracy with respect to the parameters considered that eventually contribute to the optimization of the learning algorithm

By the next approach, I will chose the best of these of three models, that can be an optimized model for the data set.
I will then use the performance metric (r2_score) and compare the three potential based on their scores, the model which has the best r2_score will be eventually considered for further analysis.
Eventually I'll optimize the selected model by 'GridSearchCV' and evaluate the model by comparing the final r2_score of the optimized model and the benchmark model.

## Benchmark

**DEFINITION** :-

A Bench Mark Model can be defined as a standard model that already shows a better performance on a given data.The factors on which our results or the solution is tested, are mostly going to be the amount of training/testing data, and then we compare your solution with that of the benchmarked solution obviously based on a performance metric(here r2_score).

The main theme here is to understand which model works delivers the best solution than their existing solution.So, it can be achieved by sheer analysis, implementing standard algorithms and observance and coming to the conclusion that the model shows good solutions or results than the benchmark model's solution.

Since, the problem is a 'Regression' task , I'm implementing a 'Linear Regression' model as my BenchMark Model.

# II. Methodology

## Data Preprocessing

**Preparing the Data**

Before we use the data as input for our machine learning algorithms(models), it has to be cleaned, formatted, and restructured — this process is known as **preprocessing**. In our dataset we have some null (NaN) values are there which are called missing values. So I will clean them in later steps. And, there are some qualities about certain features that must be adjusted. This preprocessing can help us to get the better outcome and predictive power of nearly all learning algorithms.

For this problem , I checked if there is is any null or NaN values.

```
In [190]: #Preprocessing data...
          print("NaN values in Target variable: ",life_exp.isnull().sum())
          features.isnull().sum()

          NaN values in Target variable:  10

Out[190]: Country                          0
          Year                             0
          Status                           0
          Adult Mortality                 10
          infant_deaths                    0
          Alcohol                        194
          percentage_expenditure           0
          Hepatitis B                    553
          Measles                          0
          BMI                             34
          under_five_deaths                0
          Polio                           19
          Total_expenditure              226
          Diphtheria                      19
          HIV/AIDS                         0
          GDP                            448
          Population                     652
          thinness_1-19_years             34
          thinness_5-9_years              34
          Income_composition_of_resources    167
          dtype: int64
```

The above picture shows that there are 10 NaN values in target variable and rest are of feature variables.
Then I ran the following code to remove NaN values and replace them with mean values of each feature containing Nan values.

```
In [191]: # from sklearn.preprocessing import Imputer as imp
          # imp = Imputer(missing_values='nan', strategy='mean', axis=0)
          # imp = imp.fit(life_exp)
          life_exp=life_exp.fillna(life_exp.mean())
          features['Status']=features['Status'].replace('Developing',0)
          features['Status']=features['Status'].replace('Developed',1)
          features['Adult Mortality']=features['Adult Mortality'].fillna(features['Adult Mortality'].mean())
          features['Alcohol']=features['Alcohol'].fillna(features['Alcohol'].mean())
          features['Hepatitis B']=features['Hepatitis B'].fillna(features['Hepatitis B'].mean())
          features['BMI']=features['BMI'].fillna(features['BMI'].mean())
          features['Polio']=features['Polio'].fillna(features['Polio'].mean())
          features['Total_expenditure']=features['Total_expenditure'].fillna(features['Total_expenditure'].mean())
          features['GDP']=features['GDP'].fillna(features['GDP'].mean())
          features['Population']=features['Population'].fillna(features['Population'].mean())
          features['Diphtheria']=features['Diphtheria'].fillna(features['Diphtheria'].mean())
          features['thinness_1-19_years']=features['thinness_1-19_years'].fillna(features['thinness_1-19_years'].mean())
          features['thinness_5-9_years']=features['thinness_5-9_years'].fillna(features['thinness_5-9_years'].mean())
          features['Income_composition_of_resources']=features['Income_composition_of_resources'].fillna(features['Income_composition_of_re
```

- By running above code all the nan values are replaced with thier mea values.

**Normalizing Numerical Features**

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as `'Population'` , `'Measles'`, `'HIV/AIDS'` above); however, normalization ensures that each feature is treated equally when applying

supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

Run the code cell below to normalize each numerical feature. We will use [`sklearn.preprocessing.MinMaxScaler`](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html) for this.

# Implementation

For the implementation, I used all three model that I took and fit them to the data which was splitted using test_train_split() method.
First Model: Decision Tree Regressor :

In the further section of the project, I'll intuitively select the best out of the three models that I considered for the current problem by using the performance metric(r2_score), based on the results generated, I'll decide best of the three models, which is optimal for the given problem.

**INITIAL MODEL EVALUATION :-**

In this section, I'll clearly show the coding implementation of the three supervised learning models

Import the necessary libraries and intialize the models and store them in respective variables. And finally comparing the r2_scores of the three learning models and decide which one is the best.

**1. Decision Tree Regressor**

In [209]:
```
# import necessary library
from sklearn.tree import DecisionTreeRegressor

# Create a Decision Tree Regressor Object
dec_reg = DecisionTreeRegressor(random_state=42)

# Fitting the model using training sets
dec_reg.fit(X_train,y_train)

# Making predictions using the testing set
dec_pred =  dec_reg.predict(X_test)

# Calculating the performance of the decision tree regressor model
dec_score = performance_metric(y_test,dec_pred)
print("Model has a coefficient of determination, R^2, of {:.2f}.".format(dec_score))
```

```
Model has a coefficient of determination, R^2, of 0.91.
```

**Observation:** This model has an a coefficient of determination, R^2 , of 0.91 and considered as good fit model.

This model got 0.91 R^2 score.

Second Model: Support Vector Machine

**2. Support Vector Regressor**

In [210]:
```
# import necessary library
from sklearn.svm import SVR

# Create a Support Vector Regressor Object
svr_reg = SVR()

# Fitting the model using training sets
svr_reg.fit(X_train,y_train)

# Making predictions using the testing set
svr_pred =  svr_reg.predict(X_test)

# Calculating the performance of the decision tree regressor model
svr_score = performance_metric(y_test,svr_pred)
print("Model has a coefficient of determination, R^2, of {:.2f}.".format(svr_score))
```

```
Model has a coefficient of determination, R^2, of -0.08.
```

This model got coefficient of determination of -0.08.

Third Model: Random Forest Regression

**3. Random Forest Regressor**

In [211]:
```
# import necessary library
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor Object
rfr_reg = RandomForestRegressor(random_state=42)

# Fitting the model using training sets
rfr_reg.fit(X_train,y_train)

# Making predictions using the testing set
rfr_pred =  rfr_reg.predict(X_test)

# Calculating the performance of the decision tree regressor model
rfr_score = performance_metric(y_test,rfr_pred)
#plt.scatter(y_test,y_pred)
#plt.show()
print("Model has a coefficient of determination, R^2, of {:.2f}.".format(rfr_score))
```

```
Model has a coefficient of determination, R^2, of 0.96.
```

This model got coefficient of determination, R^2 of 0.96.
Comparing the three models , Random forest regressor got an excellent coefficient determination , R^2 of 0.96. Hence, I preferred this model which is having high R^2 score , also good fit to the data. By using this model I can predict the life expectancy accurately.

# REFINEMENT

In this section of the project, the model('Random Forest Regressor Model') is optimized by the application of 'GridSearchCV' technique for fine tuning the parameters for the final model thus choosen and later calculating the performance metric(r2_score) of the optimized model.

# MODEL TUNING

Here, I will find the implementaion of GridSearchCV by intially importing the libraries sklearn.grid_search.GridSearchCV and sklearn.metrics.make_scorer.

1. Initialize the regressor('Random Forest Regressor Model') and store it in the varaible 'lgr_grid'.

2. Creating a dictionary of parameters, in the variable parameters.

3. Using make_scorer to create a r2_score scoring object. -> scorer

4. Perform Grid Search on the Regressor lgr_grid using the 'scorer', and store it in grid_obj.

5. Fit the Grid Search Object to the training data (X_train, y_train) and store it in the grid_fit.

In [212]:
```
# importing necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV


def fit_model(X_train,y_train):
    # Initialize the Regressor
    lgr_grid = RandomForestRegressor(random_state=42)

    # Create the parameters list to tune.
    params = {"n_estimators"      : [105,205,305],
              "max_features"      : ["auto", "sqrt", "log2"],
              "min_samples_split" : [2,4,8],
              "bootstrap": [True, False]}

    # Make an r2_scorer scoring object using make_scorer()
    scorer = make_scorer(performance_metric)

    # Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
    grid_obj = GridSearchCV(lgr_grid,params,scoring=scorer)

    # Fit the grid search object to the training data and find the optimal parameters using fit()
    grid_fit = grid_obj.fit(X_train,y_train)

    # calculate the estimator
    best_clf = grid_fit.best_estimator_

    # Return the optimal model after fitting the data
    best_predictions = best_clf.predict(X_test)

    return grid_fit.best_estimator_
```

In [213]:
```
rf_reg = fit_model(X_train, y_train)
rf_pred = rf_reg.predict(X_test)
print("regression score of {:.3f}".format(performance_metric(y_test,rf_pred)))
# Produce the value for 'max_depth'
#print("Parameter 'max_depth' is {} for the optimal model.".format(reg.get_params()['max_depth'])
```

```
regression score of 0.967
```

**Observation :-**

- It is quite obvious that the 'Random Forest Regressor' Model showed a best improvement upon Model Tuning using the GridSearchCV technique.
- We can observe the trend of the improvement by the values tabulated as follows :-

| Metic | Random Forest Regressor(Before Tuning) | Random Forest Regressor Model(After Tuning) |
|---|---|---|
| r2 score | 0.96 | 0.967 |

It is quite evident from the table that the Random Forest Regressor showed a good improvement upon optimization by the target variable accounting for about 96.7% of the variance.

# IV. Results

## Model Evaluation and Validation

In this part of the project I'll demonstrate the comparison of the performances between the BenchMark Model('Linear Regressor') and the Opmital Model('Random Tree Regressor') based on their performance metrics(r2_score) in a tabular form.

**Results**

| **Metric** | **Linear Regressor Model(BenchMark Model)** | **Random Forest Regressor Model(Optimal Model)** |
|---|---|---|
| r2 score | 0.80 | 0.967 |

## Observation :-

Although the performances of the BenchMark Model and the Optimal Model thus considered are quite figurative and are literally showing nearly similar performances, it is quite evident that the Random Forest Regressor Model(Optimal Model) shows an awesome performance than the Linear Regressor Model(BenchMark Model).

And it is quite obvious that Random Forest Regressor Model i.e., Optimal Model shows a better performance on the input dataset.

# MODEL VALIDATION

In this part of the project, I'll demonstrate the performance of the Best Model for the given regression task i.e., The Optimal Model aganist unseen data.

**Task - Predicting Life expectancy of some countries**

Here I'll take some countries as examples to predict the life expectancy by taking all the necessary data into account. I wanna take 3 countries data to predict life expectancy. Below cell will produce data of a 3 countries, probably U.S.A, Japan and India. We will store it in grid or matrix.

In [217]:
```
features.loc[[2794,1314,1186]]
```

Out[217]:

| | Country | Year | Status | Adult Mortality | infant_deaths | Alcohol | percentage_expenditure | Hepatitis B | Measles | BMI | under_five_deaths | Polio | Total_expenditure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2794** | 175 | 2015 | 1 | 13.0 | 23 | 4.602861 | 0.0 | 92.000000 | 0.000886 | 69.6 | 26 | 93.0 | 5.93819 |
| **1314** | 83 | 2015 | 1 | 55.0 | 2 | 4.602861 | 0.0 | 80.940461 | 0.000165 | 29.0 | 3 | 99.0 | 5.93819 |
| **1186** | 75 | 2015 | 0 | 181.0 | 910 | 4.602861 | 0.0 | 87.000000 | 0.425986 | 18.7 | 1100 | 86.0 | 5.93819 |

In [218]:
```
# Following is the code for predicting the life expectancy by considering the Random Forest Regressor Model i.e.,The Optimal Mode
#k_scores = reg.feature_importances_
# Producing a matrix for the the data
my_data = [[175,2015,1,0.016620,23,4.602861,0.0,92.000000,0.000886,0.794902,26,93.0,5.93819,95.0,0.00000,7483.158469,0.009857,0.8
                [83,2015,1,0.074792,2,4.602861,0.0,80.940461,0.000165,0.324450,3,99.0,5.93819,96.0,0.00000,34474.137360,0.000098,
                [75,2015,0,0.249307,910,4.602861,0.0,87.000000,0.425986,0.205098,1100,86.0,5.93819,87.0,0.00198,1613.188780,0.001
# Displaying the results
#print(k_scores)
#print student_data
#print reg.predict(student_data)
#plt.plot(reg.predict(student_data),student_data)
#plt.show()
for i, score in enumerate(rf_reg.predict(my_data)):
    print("Predicted life expectancy of country {}: {:,.2f}".format(i+1, score))
```

```
Predicted life expectancy of country 1: 77.93
Predicted life expectancy of country 2: 83.71
Predicted life expectancy of country 3: 68.91
```

## Justification

By observing the validation results above, it is quite evident that the model is performing well on the given data.
When compared to the BenchMark Model, the Optimal Model('Random Forest Regressor Model') indeed shows a striking performance as shown in the 'Final Model Evaluation' section.

- Although the performance scores are so close and similar, it is likely to understand that the Optimal Model('Random Forest Regressor Model') holds a tight control in generalizing the input data.
- The performance of the BenchMark Model is 80.02 % and that of the Optimal Model is 96.7%.

The model can be applied to predict the life expectancy of people in a country.

# V. Conclusion

## Reflection

**The process used for this project can be summarized using the following steps:-**

1. A Common Problem on Life Expectancy is found in "Kaggle" and the data set related to it is acquired from the Public Domain.

2. The Dataset was downloaded and loaded for the current project and necessary statistics were calculated.
3. The Data Set is Explored with scatter-plots to illustrate the correlation of the input features with respect to the target variable.
4. The Data Set is preprocessed using the technique "Feature Selection" and irrelevant attributes are removed from the dataset.
5. A function is designed that calculates the performance metric r2_score and returns that score.
6. The data is splitted into 80% training and 20% testing data.
7. A BenchMark was created for the Regressor i.e, here Linear Regressor acts as a BenchMark Model and is trained using the training data.
8. Then three supervised learning models were trained using the training data and a comparision is done based on the performance metric and decided which among the three is the best model.
9. The best model thus selected is optimized by the application of 'GridSearchCV'.
10. The optimized model is then compared with the BenchMark Model and came to the conclusion that the Optimzed Model shows a better performance than the BenchMark model.
11. Finally the Optimized Model which is selected as the best model for the input data is validated aganist unseen data and the performance and intuition was documented clearly.

- I personally found the steps 3 and 4 most challenging because I have to completely understand my data and draw relevant intuition from it. Then I found that all the features are important in predicting life expectancy of people in a country.

# Improvement

Potentially the 'Data Preprocessing' phase is the crucial part of any Machine Learning Problem, Since during this phase we can potentially identify the flaws in the data set that could actually mess the results and performance of the model thus considered.

- Hence, removing irrelevant data during this phase can predominantly increase the model's performance and can benefit in generalized results.
- But I personally feel that the feature selection that I used is good but there are other techniques that are used for the application of Feature Selection.
- There is also room for trying Embedded method such as LASSO and Elastic Net and Ridge Regression that don't need the external implementation of the feature selection techniques , since these methods have embedded feature selection and regularization built in. It's worth a trial, since there is always scope for improving the model performance aganist the given dataset.
- Further more, some ensemble methods such as 'XGBOOST' should also take care of larger data dimensions and these methods can themselves be used for feature selection.

There are lot of other possibilites that can make the feature selection more intuitive and literally I am new to this and the 'Feature Selection' implementation gave me a hard core challenging of applying it in real time.