ÖZYEĞİN UNIVERSITY

## *CS 452*
## *Data Science with Python*
## *Assignment 1 Report*

# *Life Expectancy by Linear Regression*



| | |
|---|---|
| ***Author*** | *Uygar KAYA* |
| ***Instructor*** | *Assistant Prof. Reyhan Aydoğan* |
| ***Submission Date*** | *14.11.2021* |

# 1 Introduction

*In this assignment, we implement a Linear Regression model that solves the problem of finding the life expectancy factor in different countries.*

*In this assignment, by pre-processing the .csv extension data file given to us, different Linear Regression models were developed to predict the life expectancy of people in different countries, including or excluding some set of features data provided to us, and these developed models were evaluated qualitatively and quantitatively in this report.*

## 1.1 Linear Regression

*Linear Regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an independent variable (y), and the other is considered to be a dependent variable (x).*

*A Linear Regression line has an equation of the form y = mx + n, where x is the independent variable and y is the dependent variable. The slope of the line is m, and n is the intercept.*

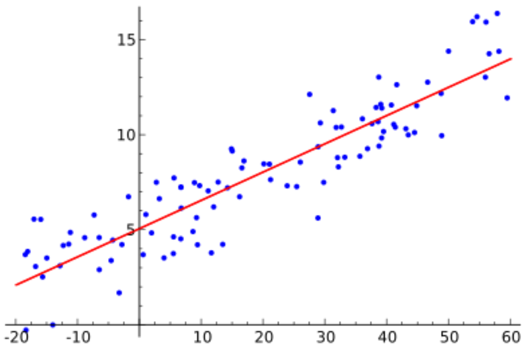



*Figure 1 - Example of Linear Regression*

# 2 Methodology

*In this assignment, we apply a Linear Regression model methodology using Python Programming Language.*

*During the these assigment, using only 5 different Python programming language libraries these are;*

1. *NumPy*
2. *Pandas*
3. *Scikit-Learn*
4. *Matplotlib*
5. *Seaborn were used.*

*First of all, the file with the .csv extension was converted into a DataFrame, then the raw data was visualized and information about the dataset was given. In addition, Categorical & Numerical features were determined in these raw data, and pre-processing was performed for these determined data. After the data was pre-processed, Dropping & Imputation (Handling with Missing Values) was performed for null values. Finally, Linear Regression model was applied after the Data Splitting into two different veriables as Test Data and Train Data. In order to find the best linear regression model, we choose the linear regression model that performs the best out of 5 different linear regression models with at least 8 features and the performance of this model is visualized & presented in this report.*

## 3 Implementation Details

*In this assigment, firstly we read the csv file with the help of the pandas library in Python and this DataFrame has 2938 rows & 22 columns in total. Also, there is no duplicate rows.*

```
# Read data from a CSV file

df = pd.read_csv('assignment-1-data.csv')
```

```
df.shape
```

```
(2938, 22)
```

*When we look at the first 5 rows with the help of the head() method, we get below table.*

```
df.head()
```

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 33736494.0 | 17.2 | 17.3 | 0.479 |
| **1** | Afghanistan | 2014 | Developing | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 492 | 18.6 | 86 | 58.0 | 8.18 | 62.0 | 0.1 | 612.696514 | 327582.0 | 17.5 | 17.5 | 0.476 |
| **2** | 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 | 31731688.0 | 17.7 | 17.7 | 0.470 |
| **3** | 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 | 3696958.0 | 17.9 | 18.0 | 0.463 |
| **4** | 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 | 2978599.0 | 18.2 | 18.2 | 0.454 |

*There are 22 different features in total, some of them are of Int & Float data types, while others are of Object data type.*

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Country                          2938 non-null   object
 1   Year                             2938 non-null   int64
 2   Status                           2938 non-null   object
 3   Life expectancy                  2928 non-null   float64
 4   Adult Mortality                  2928 non-null   float64
 5   infant deaths                    2938 non-null   int64
 6   Alcohol                          2744 non-null   float64
 7   percentage expenditure           2938 non-null   float64
 8   Hepatitis B                      2385 non-null   float64
 9   Measles                          2938 non-null   int64
 10   BMI                             2904 non-null   float64
 11  under-five deaths                2938 non-null   int64
 12  Polio                            2919 non-null   float64
 13  Total expenditure                2712 non-null   float64
 14  Diphtheria                       2919 non-null   float64
 15   HIV/AIDS                        2938 non-null   float64
 16  GDP                              2490 non-null   float64
 17  Population                       2286 non-null   float64
 18   thinness  1-19 years            2904 non-null   float64
 19   thinness 5-9 years              2904 non-null   float64
 20  Income composition of resources  2771 non-null   float64
 21  Schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 527.9+ KB
```

*There are 16 float features which are, Life Expectancy, Adult Mortality, Alcohol, Percentage Expenditure, Hepatitis B, BMI, Polio, Total Expenditure, Diphtheria, HIV/AIDS, GDP, Population, Thinness 1-19 years, Thinness 5-9 years, Income Composition of Resources and Schooling, 4 int features which are Year, Infant Deaths, Measles and Under-Five Deaths. Also, 2 object features which are Country & Status. Country & Status features are Categorical data and the rest of them are Numerical data.*

```
# Summary of Numeric Features

df.describe().round(1)
```

| | Year | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | Measles | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | Population | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2938.0 | 2928.0 | 2928.0 | 2938.0 | 2744.0 | 2938.0 | 2385.0 | 2938.0 | 2904.0 | 2938.0 | 2919.0 | 2712.0 | 2919.0 | 2938.0 | 2490.0 | 2.286000e+03 | 2904.0 | 2904.0 | 2771.0 | 2775.0 |
| **mean** | 2007.5 | 69.2 | 164.8 | 30.3 | 4.6 | 738.3 | 80.9 | 2419.6 | 38.3 | 42.0 | 82.6 | 5.9 | 82.3 | 1.7 | 7483.2 | 1.275338e+07 | 4.8 | 4.9 | 0.6 | 12.0 |
| **std** | 4.6 | 9.5 | 124.3 | 117.9 | 4.1 | 1987.9 | 25.1 | 11467.3 | 20.0 | 160.4 | 23.4 | 2.5 | 23.7 | 5.1 | 14270.2 | 6.101210e+07 | 4.4 | 4.5 | 0.2 | 3.4 |
| **min** | 2000.0 | 36.3 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 3.0 | 0.4 | 2.0 | 0.1 | 1.7 | 3.400000e+01 | 0.1 | 0.1 | 0.0 | 0.0 |
| **25%** | 2004.0 | 63.1 | 74.0 | 0.0 | 0.9 | 4.7 | 77.0 | 0.0 | 19.3 | 0.0 | 78.0 | 4.3 | 78.0 | 0.1 | 463.9 | 1.957932e+05 | 1.6 | 1.5 | 0.5 | 10.1 |
| **50%** | 2008.0 | 72.1 | 144.0 | 3.0 | 3.8 | 64.9 | 92.0 | 17.0 | 43.5 | 4.0 | 93.0 | 5.8 | 93.0 | 0.1 | 1766.9 | 1.386542e+06 | 3.3 | 3.3 | 0.7 | 12.3 |
| **75%** | 2012.0 | 75.7 | 228.0 | 22.0 | 7.7 | 441.5 | 97.0 | 360.2 | 56.2 | 28.0 | 97.0 | 7.5 | 97.0 | 0.8 | 5910.8 | 7.420359e+06 | 7.2 | 7.2 | 0.8 | 14.3 |
| **max** | 2015.0 | 89.0 | 723.0 | 1800.0 | 17.9 | 19479.9 | 99.0 | 212183.0 | 87.3 | 2500.0 | 99.0 | 17.6 | 99.0 | 50.6 | 119172.7 | 1.293859e+09 | 27.7 | 28.6 | 0.9 | 20.7 |

*There are number of missing rows in the some features.*

```
# Check the rows with missing values

df.isnull().sum()
```

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

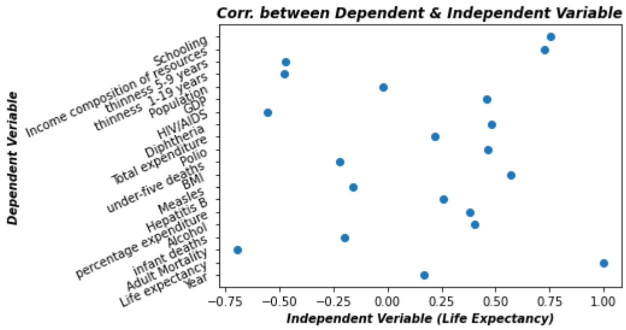
*The most NaN values are available in the Population, Hepatitis B and GDP columns. In the pre-processing part, the correlation of these NaN values with the independent variable checked and the columns with a very weak correlation relationship dropped. The remaining NaN values were processed with the Handling with Missing Values (Dropping & Imputation) method and used in the Linear Regression model by generating data for the missing numerical values.*

*Before applying the pre-processing part, the correlation relationship between the independent variables and numerical dependent variables was analyzed.*

```
# Applying the Pearson Correlation to find the Correlation between features
df.corr(method="pearson")
```

| | Year | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | Measles | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | Population | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Year** | 1.000000 | 0.170033 | -0.079052 | -0.037415 | -0.052990 | 0.031400 | 0.104333 | -0.082493 | 0.108974 | -0.042937 | 0.094158 | 0.090740 | 0.134337 | -0.139741 | 0.101620 | 0.016969 | -0.047876 | -0.050929 | 0.243468 | 0.209400 |
| **Life expectancy** | 0.170033 | 1.000000 | -0.696359 | -0.196557 | 0.404877 | 0.381864 | 0.256762 | -0.157586 | 0.567694 | -0.222529 | 0.465556 | 0.218086 | 0.479495 | -0.556556 | 0.461455 | -0.021538 | -0.477183 | -0.471584 | 0.724776 | 0.751975 |

```
# Visualizing the Correlation Between Dependent and Independent Variable
independentVarCorr = df.corr(method="pearson").iloc[1, :]

plt.xlabel("Independent Veriable (Life Expectancy)", fontweight="bold", style="italic")
plt.ylabel("Dependent Veriable", fontweight="bold", style="italic")
plt.yticks(rotation = 25)

plt.title("Corr. between Dependent & Independent Variable", fontweight="bold", style="italic")
plt.scatter(independentVarCorr, ["Year", "Life expectancy", "Adult Mortality", "infant deaths", "Alcohol",
                                 "percentage expenditure", "Hepatitis B", "Measles ", " BMI ", "under-five deaths ", "Polio",
                                 "Total expenditure", "Diphtheria ", " HIV/AIDS", "GDP", "Population", " thinness  1-19 years",
                                 " thinness 5-9 years", "Income composition of resources", "Schooling"])
plt.show()
```




*Features with a correlation relationship between independent and dependent variables of -0.2 < x < 0.2 will not affect the accuracy of the Linear Regression model by dropping these features. Thus, I dropped the Year (0.170033), Measles (-0.157586), and Population (-0.021538) columns.*

## *3.1 Data Pre-Processing Part*

### *3.1.1 Droping Very Weak Correlation Data*

*After specifically dropping 4 columns, the following table is formed.*

```
# Drop the 'Year', 'Measles' & 'Population' because it has very low Correlation to do with the Independent Variable
# Drop the 'Country' because just it's an illusion
df.drop(columns=['Country', 'Year', 'Measles ', 'Population'], inplace = True)
```

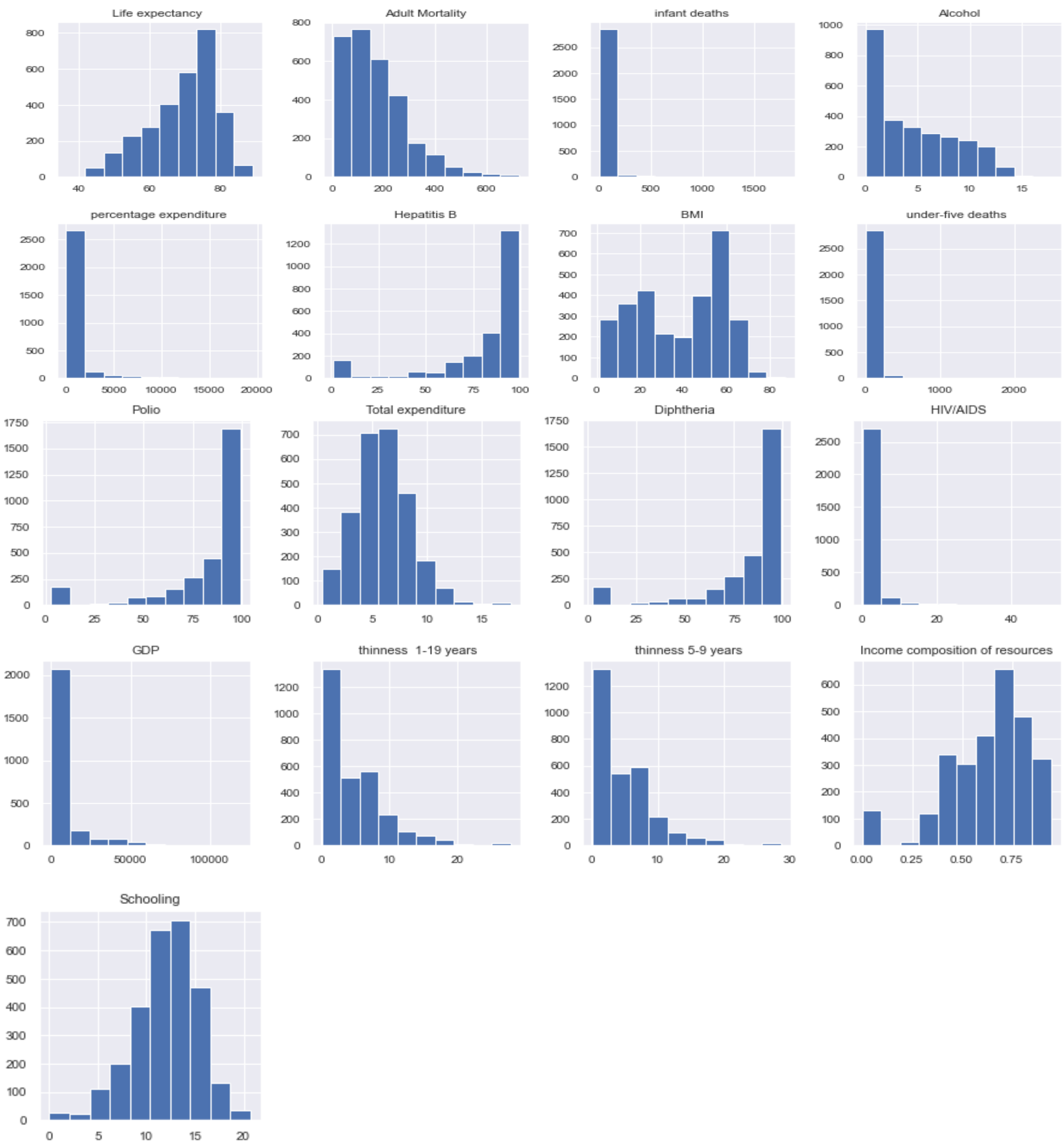
```
df
```

| | Status | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Developing | 65.0 | 263.0 | 62 | 0.01 | 71.279624 | 65.0 | 19.1 | 83 | 6.0 | 8.16 | 65.0 | 0.1 | 584.259210 | 17.2 | 17.3 | 0.479 | 10.1 |
| **1** | Developing | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 18.6 | 86 | 58.0 | 8.18 | 62.0 | 0.1 | 612.696514 | 17.5 | 17.5 | 0.476 | 10.0 |
| **2** | Developing | 59.9 | 268.0 | 66 | 0.01 | 73.219243 | 64.0 | 18.1 | 89 | 62.0 | 8.13 | 64.0 | 0.1 | 631.744976 | 17.7 | 17.7 | 0.470 | 9.9 |
| **3** | Developing | 59.5 | 272.0 | 69 | 0.01 | 78.184215 | 67.0 | 17.6 | 93 | 67.0 | 8.52 | 67.0 | 0.1 | 669.959000 | 17.9 | 18.0 | 0.463 | 9.8 |
| **4** | Developing | 59.2 | 275.0 | 71 | 0.01 | 7.097109 | 68.0 | 17.2 | 97 | 68.0 | 7.87 | 68.0 | 0.1 | 63.537231 | 18.2 | 18.2 | 0.454 | 9.5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2933** | Developing | 44.3 | 723.0 | 27 | 4.36 | 0.000000 | 68.0 | 27.1 | 42 | 67.0 | 7.13 | 65.0 | 33.6 | 454.366654 | 9.4 | 9.4 | 0.407 | 9.2 |
| **2934** | Developing | 44.5 | 715.0 | 26 | 4.06 | 0.000000 | 7.0 | 26.7 | 41 | 7.0 | 6.52 | 68.0 | 36.7 | 453.351155 | 9.8 | 9.9 | 0.418 | 9.5 |
| **2935** | Developing | 44.8 | 73.0 | 25 | 4.43 | 0.000000 | 73.0 | 26.3 | 40 | 73.0 | 6.53 | 71.0 | 39.8 | 57.348340 | 1.2 | 1.3 | 0.427 | 10.0 |
| **2936** | Developing | 45.3 | 686.0 | 25 | 1.72 | 0.000000 | 76.0 | 25.9 | 39 | 76.0 | 6.16 | 75.0 | 42.1 | 548.587312 | 1.6 | 1.7 | 0.427 | 9.8 |
| **2937** | Developing | 46.0 | 665.0 | 24 | 1.68 | 0.000000 | 79.0 | 25.5 | 39 | 78.0 | 7.10 | 78.0 | 43.5 | 547.358878 | 11.0 | 11.2 | 0.434 | 9.8 |

2938 rows × 18 columns

*I suppressed the histogram for the numeric values after the unnecessary columns that we will not use in the linear regression model were dropped.*

```
# Drawing a histogram for each feature
df.hist(figsize=(15,20))

# Clear the text "residue"
plt.show()
```



### *3.1.2 Dealing with Missing Values (Dropping & Imputation)*

*Instead of directly using dropna() method for the missing value, I take the mean of them and replacing them with NaN value has ensured that we do not experience a decrease in the number of rows. If we used dropna(), all values with NaN rows would be lost, which would have 1649 rows of data instead of 2938 rows data, but this data would reduce the accuracy of our model. Therefore, we can better train our model by filling in NaN values without reducing our row count by using the imputation methodology.*

```
# Dealing with missing values and replacing them by their mean

df["Life expectancy "].fillna(df["Life expectancy "].mean(), inplace=True)
df["Adult Mortality"].fillna(df["Adult Mortality"].mean(), inplace=True)
df["Alcohol"].fillna(df["Alcohol"].mean(), inplace=True)
df["Hepatitis B"].fillna(df["Hepatitis B"].mean(), inplace=True)
df[" BMI "].fillna(df[" BMI "].mean(), inplace=True)
df["Polio"].fillna(df["Polio"].mean(), inplace=True)
df["Total expenditure"].fillna(df["Total expenditure"].mean(), inplace=True)
df["Diphtheria "].fillna(df["Diphtheria "].mean(), inplace=True)
df["GDP"].fillna(df["GDP"].mean(), inplace=True)
df[" thinness  1-19 years"].fillna(df[" thinness  1-19 years"].mean(), inplace=True)
df[" thinness 5-9 years"].fillna(df[" thinness 5-9 years"].mean(), inplace=True)
df["Income composition of resources"].fillna(df["Income composition of resources"].mean(), inplace=True)
df["Schooling"].fillna(df["Schooling"].mean(), inplace=True)
```

*After these pre-processing parts, we have one Categorical data which is Status. I applied the One-Hot Encoding methodology to convert the status data which is the Categorical data, into a numeric value as well.*

### *3.1.3 Handling Categorical Data (One-Hot Encoding)*

*After applying the get_dummies() method, the following table is formed*

```
# applying one-hot encoding

df = pd.get_dummies(df[['Status','Life expectancy ','Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure',
                    'Hepatitis B', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure', 'Diphtheria ', ' HIV/AIDS',
                    'GDP', ' thinness  1-19 years', ' thinness 5-9 years', 'Income composition of resources', 'Schooling']])
```

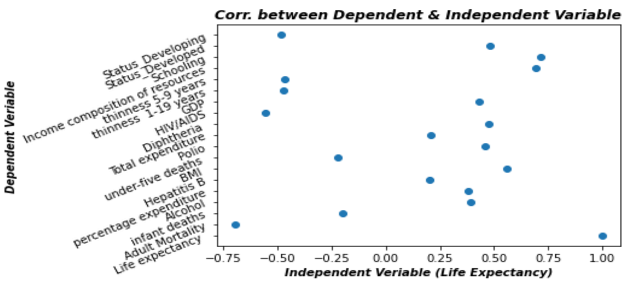
| | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling | Status_Developed | Status_Developing |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 65.0 | 263.0 | 62 | 0.01 | 71.279624 | 65.0 | 19.1 | 83 | 6.0 | 8.16 | 65.0 | 0.1 | 584.259210 | 17.2 | 17.3 | 0.479 | 10.1 | 0 | 1 |
| 1 | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 18.6 | 86 | 58.0 | 8.18 | 62.0 | 0.1 | 612.696514 | 17.5 | 17.5 | 0.476 | 10.0 | 0 | 1 |
| 2 | 59.9 | 268.0 | 66 | 0.01 | 73.219243 | 64.0 | 18.1 | 89 | 62.0 | 8.13 | 64.0 | 0.1 | 631.744976 | 17.7 | 17.7 | 0.470 | 9.9 | 0 | 1 |
| 3 | 59.5 | 272.0 | 69 | 0.01 | 78.184215 | 67.0 | 17.6 | 93 | 67.0 | 8.52 | 67.0 | 0.1 | 669.959000 | 17.9 | 18.0 | 0.463 | 9.8 | 0 | 1 |
| 4 | 59.2 | 275.0 | 71 | 0.01 | 7.097109 | 68.0 | 17.2 | 97 | 68.0 | 7.87 | 68.0 | 0.1 | 63.537231 | 18.2 | 18.2 | 0.454 | 9.5 | 0 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2933 | 44.3 | 723.0 | 27 | 4.36 | 0.000000 | 68.0 | 27.1 | 42 | 67.0 | 7.13 | 65.0 | 33.6 | 454.366654 | 9.4 | 9.4 | 0.407 | 9.2 | 0 | 1 |
| 2934 | 44.5 | 715.0 | 26 | 4.06 | 0.000000 | 7.0 | 26.7 | 41 | 7.0 | 6.52 | 68.0 | 36.7 | 453.351155 | 9.8 | 9.9 | 0.418 | 9.5 | 0 | 1 |
| 2935 | 44.8 | 73.0 | 25 | 4.43 | 0.000000 | 73.0 | 26.3 | 40 | 73.0 | 6.53 | 71.0 | 39.8 | 57.348340 | 1.2 | 1.3 | 0.427 | 10.0 | 0 | 1 |
| 2936 | 45.3 | 686.0 | 25 | 1.72 | 0.000000 | 76.0 | 25.9 | 39 | 76.0 | 6.16 | 75.0 | 42.1 | 548.587312 | 1.6 | 1.7 | 0.427 | 9.8 | 0 | 1 |
| 2937 | 46.0 | 665.0 | 24 | 1.68 | 0.000000 | 79.0 | 25.5 | 39 | 78.0 | 7.10 | 78.0 | 43.5 | 547.358878 | 11.0 | 11.2 | 0.434 | 9.8 | 0 | 1 |

*Correlation information was analyzed again for the new data I obtained after One-Hot Encoding. There is very little change in the correlation values with the new data created after taking the mean, but the correlation with these changes did not go below the weak correlation.*

```
# Applying the Pearson Correlation to find the Correlation between new features
df.corr(method="pearson")
```

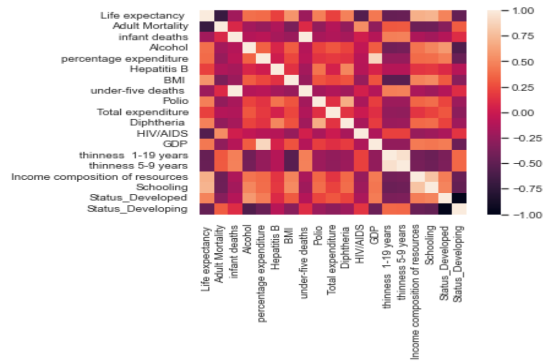
| | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling | Status_Developed | Status_Developing |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Life expectancy | 1.000000 | -0.696359 | -0.196535 | 0.391598 | 0.381791 | 0.203771 | 0.559255 | -0.222503 | 0.461574 | 0.207981 | 0.475418 | -0.556457 | 0.430493 | -0.472162 | -0.466629 | 0.692483 | 0.715066 | 0.481962 | -0.481962 |

```
# Visualizing the New Correlation Between Dependent and Independent Variable
independentVarCorrNew = df.corr(method="pearson").iloc[0, :]

plt.xlabel("Independent Veriable (Life Expectancy)", fontweight="bold", style="italic")
plt.ylabel("Dependent Veriable", fontweight="bold", style="italic")
plt.yticks(rotation = 25)

plt.title("Corr. between Dependent & Independent Variable", fontweight="bold", style="italic")
plt.scatter(independentVarCorrNew, ["Life expectancy ", "Adult Mortality", "infant deaths", "Alcohol", "percentage expenditure",
                                    "Hepatitis B", " BMI ", "under-five deaths ", "Polio", "Total expenditure", "Diphtheria ",
                                    " HIV/AIDS", "GDP", " thinness  1-19 years", " thinness 5-9 years",
                                    "Income composition of resources", "Schooling", "Status_Developed", "Status_Developing"])
plt.show()
```

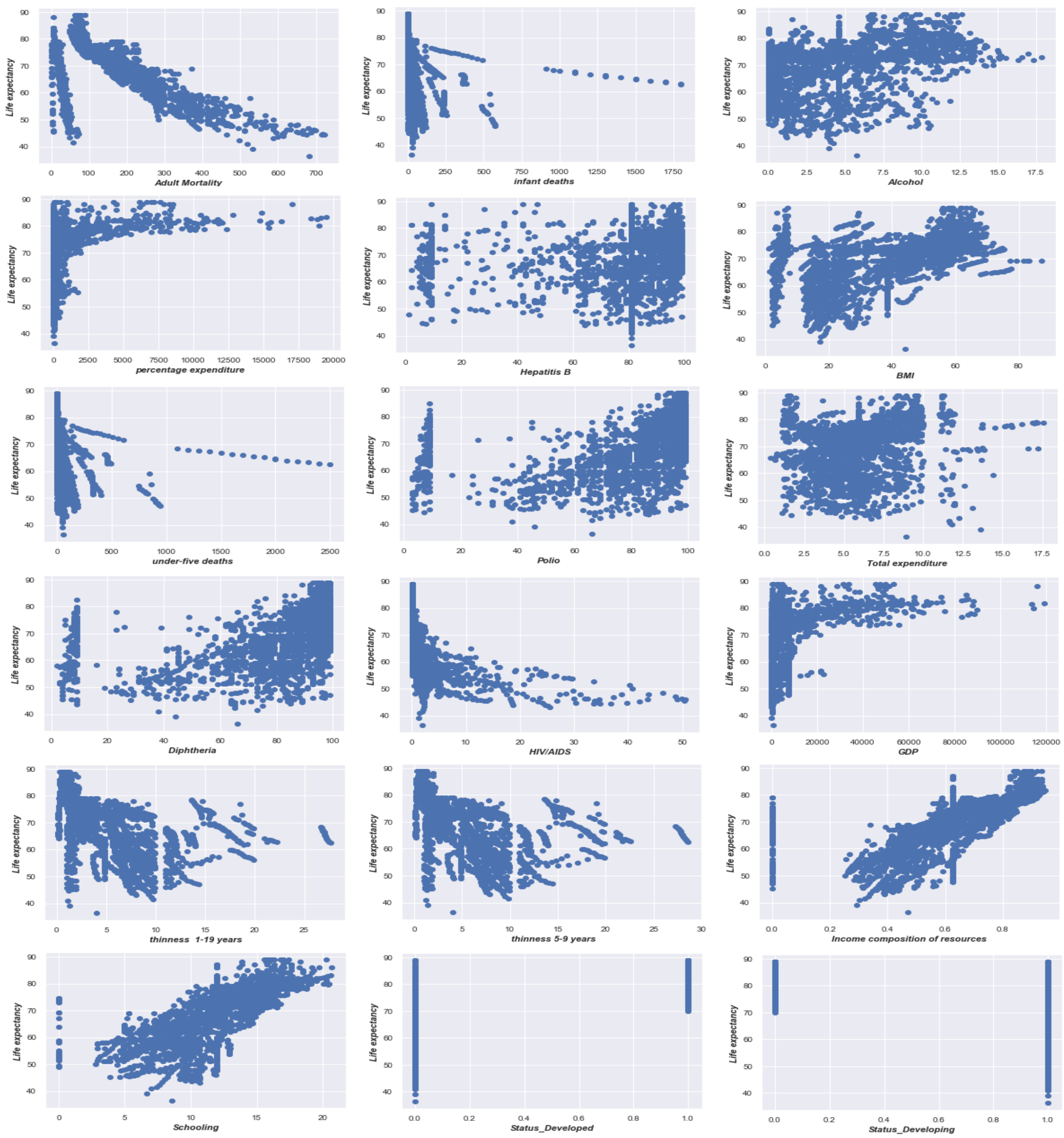


```
#Draw a heat map

sns.set(font_scale=0.9)
sns.heatmap(df.corr(method="pearson"), cbar=True, square=True)
```

```
<AxesSubplot:>
```




*Visualizing the Data for each Dependent Feature with Independent Feature after Pre-Processing and One-Hot Encoding*



*If I hadn't done any statistical analysis such as correlation, the columns I would choose for my model would be "Income composition of resources", "Schooling", "Diphtheria", "HIV/AIDS", "BMI", "Alcohol" and "Adult Mortality" because educated people have a certain level of knowledge to live healthily. That's why I think people who live healthily will have a longer Life Expectancy, so I choose the "Schooling". I chose "Diphtheria" & "HIV/AIDS" columns because these diseases are conditions that reduce a person's life expectancy. Other selected columns, I think it has an effect on Life Expectancy.*

*In this assigment, an experimental environment was created by creating 5 different models and using at least 8 columns for each model to see that different columns would produce different results for the models. That's why I ranked the correlation values for the 5 different DataFrames I'm going to create, from highest to lowest.*

```
# Sorting the correlations between Dependent & Independent Veriable
df.corr(method="pearson").iloc[0, :].abs().sort_values(ascending = False)
```

```
Life expectancy                    1.000000
Schooling                          0.715066
Adult Mortality                    0.696359
Income composition of resources    0.692483
 BMI                               0.559255
 HIV/AIDS                          0.556457
Status_Developing                  0.481962
Status_Developed                   0.481962
Diphtheria                         0.475418
 thinness  1-19 years              0.472162
 thinness 5-9 years                0.466629
Polio                              0.461574
GDP                                0.430493
Alcohol                            0.391598
percentage expenditure             0.381791
under-five deaths                  0.222503
Total expenditure                  0.207981
Hepatitis B                        0.203771
infant deaths                      0.196535
Name: Life expectancy , dtype: float64
```

*In df1, we use all the data what we pre-process.*

```
df1.head()
```

| | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | BMI | under-five deaths | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP | thinness 1-19 years | thinness 5-9 years | Income composition of resources | Schooling | Status_Developed | Status_Developing |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 65.0 | 263.0 | 62 | 0.01 | 71.279624 | 65.0 | 19.1 | 83 | 6.0 | 8.16 | 65.0 | 0.1 | 584.259210 | 17.2 | 17.3 | 0.479 | 10.1 | 0 | 1 |
| 1 | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 18.6 | 86 | 58.0 | 8.18 | 62.0 | 0.1 | 612.696514 | 17.5 | 17.5 | 0.476 | 10.0 | 0 | 1 |
| 2 | 59.9 | 268.0 | 66 | 0.01 | 73.219243 | 64.0 | 18.1 | 89 | 62.0 | 8.13 | 64.0 | 0.1 | 631.744976 | 17.7 | 17.7 | 0.470 | 9.9 | 0 | 1 |
| 3 | 59.5 | 272.0 | 69 | 0.01 | 78.184215 | 67.0 | 17.6 | 93 | 67.0 | 8.52 | 67.0 | 0.1 | 669.959000 | 17.9 | 18.0 | 0.463 | 9.8 | 0 | 1 |
| 4 | 59.2 | 275.0 | 71 | 0.01 | 7.097109 | 68.0 | 17.2 | 97 | 68.0 | 7.87 | 68.0 | 0.1 | 63.537231 | 18.2 | 18.2 | 0.454 | 9.5 | 0 | 1 |

*In df2, we use 8 different dependent variables data with the highest correlation with the independent variable.*

```
df2.head()
```

| | Schooling | Adult Mortality | Income composition of resources | BMI | HIV/AIDS | Status_Developing | Status_Developed | Diphtheria | Life expectancy |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 10.1 | 263.0 | 0.479 | 19.1 | 0.1 | 1 | 0 | 65.0 | 65.0 |
| 1 | 10.0 | 271.0 | 0.476 | 18.6 | 0.1 | 1 | 0 | 62.0 | 59.9 |
| 2 | 9.9 | 268.0 | 0.470 | 18.1 | 0.1 | 1 | 0 | 64.0 | 59.9 |
| 3 | 9.8 | 272.0 | 0.463 | 17.6 | 0.1 | 1 | 0 | 67.0 | 59.5 |
| 4 | 9.5 | 275.0 | 0.454 | 17.2 | 0.1 | 1 | 0 | 68.0 | 59.2 |

*In df3, we use 8 different dependent variables data with the lowest correlation with the independent variable.*

```
df3.head()
```

| | infant deaths | Hepatitis B | Total expenditure | under-five deaths | percentage expenditure | Alcohol | GDP | Polio | Life expectancy |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 62 | 65.0 | 8.16 | 83 | 71.279624 | 0.01 | 584.259210 | 6.0 | 65.0 |
| 1 | 64 | 62.0 | 8.18 | 86 | 73.523582 | 0.01 | 612.696514 | 58.0 | 59.9 |
| 2 | 66 | 64.0 | 8.13 | 89 | 73.219243 | 0.01 | 631.744976 | 62.0 | 59.9 |
| 3 | 69 | 67.0 | 8.52 | 93 | 78.184215 | 0.01 | 669.959000 | 67.0 | 59.5 |
| 4 | 71 | 68.0 | 7.87 | 97 | 7.097109 | 0.01 | 63.537231 | 68.0 | 59.2 |

*In df4, we use 10 different dependent variables data that have one lowest and one highest correlation with the independent variable.*

```
df4.head()
```

| | Schooling | infant deaths | Adult Mortality | Hepatitis B | Income composition of resources | Total expenditure | BMI | under-five deaths | HIV/AIDS | percentage expenditure | Life expectancy |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10.1 | 62 | 263.0 | 65.0 | 0.479 | 8.16 | 19.1 | 83 | 0.1 | 71.279624 | 65.0 |
| **1** | 10.0 | 64 | 271.0 | 62.0 | 0.476 | 8.18 | 18.6 | 86 | 0.1 | 73.523582 | 59.9 |
| **2** | 9.9 | 66 | 268.0 | 64.0 | 0.470 | 8.13 | 18.1 | 89 | 0.1 | 73.219243 | 59.9 |
| **3** | 9.8 | 69 | 272.0 | 67.0 | 0.463 | 8.52 | 17.6 | 93 | 0.1 | 78.184215 | 59.5 |
| **4** | 9.5 | 71 | 275.0 | 68.0 | 0.454 | 7.87 | 17.2 | 97 | 0.1 | 7.097109 | 59.2 |

*In df5, we use data with modeate and above correlation values.*

```
df5.head()
```

| | Schooling | Adult Mortality | Income composition of resources | BMI | HIV/AIDS | Status_Developing | Status_Developed | Diphtheria | thinness 1-19 years | thinness 5-9 years | Polio | GDP | Life expectancy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10.1 | 263.0 | 0.479 | 19.1 | 0.1 | 1 | 0 | 65.0 | 17.2 | 17.3 | 6.0 | 584.259210 | 65.0 |
| **1** | 10.0 | 271.0 | 0.476 | 18.6 | 0.1 | 1 | 0 | 62.0 | 17.5 | 17.5 | 58.0 | 612.696514 | 59.9 |
| **2** | 9.9 | 268.0 | 0.470 | 18.1 | 0.1 | 1 | 0 | 64.0 | 17.7 | 17.7 | 62.0 | 631.744976 | 59.9 |
| **3** | 9.8 | 272.0 | 0.463 | 17.6 | 0.1 | 1 | 0 | 67.0 | 17.9 | 18.0 | 67.0 | 669.959000 | 59.5 |
| **4** | 9.5 | 275.0 | 0.454 | 17.2 | 0.1 | 1 | 0 | 68.0 | 18.2 | 18.2 | 68.0 | 63.537231 | 59.2 |

*Before applying the Linear Regression model, I guess that the best result will be df1 or df5 because I created the data with the highest correlation value in the correlation of the value I will predict and the dependent variables in these DataFrames. Therefore, I think that one of these two models will give the best performance. Also, I think df3 will produce the worst performance because I created the data with the lowest correlation relationship in this DataFrame.*

### *3.1.4 Applying Linear Regression Model Strategy*

*First of all, we separate the data we want to predict which is independent veriable and the dependent variable that we will use in this predicted data. Also, we divide the data created in DataFrames into test and train to apply Linear Regression. The test data will make up 20% of the whole DataFrame in total, so we set the test_size to 0.2 and we set the random_state to 147.*

```
# Split the data to independent variables(y) and dependent variable(x)
# Y1, Y2, Y3, Y4, Y5 is target feature that we want to make prediction on

X1 = df1.iloc[:,1:19].values
Y1 = df1.iloc[:,0].values

X2 = df2.iloc[:,0:8].values
Y2 = df2.iloc[:,-1].values

X3 = df3.iloc[:,0:8].values
Y3 = df3.iloc[:,-1].values

X4 = df4.iloc[:,0:10].values
Y4 = df4.iloc[:,-1].values

X5 = df5.iloc[:,0:12].values
Y5 = df5.iloc[:,-1].values
```

```
# We split the data into training and test sets

X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size=0.2, random_state=147)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size=0.2, random_state=147)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(X3, Y3, test_size=0.2, random_state=147)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(X4, Y4, test_size=0.2, random_state=147)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(X5, Y5, test_size=0.2, random_state=147)
```

*Then we look at the shapes of the training sets we have created, because the shapes of the train sets we have separated must be the same.*

```
# We should look the X train & Y train data shape because it should be equal
x_train_shape = [X1_train.shape, X2_train.shape, X3_train.shape, X4_train.shape, X5_train.shape]
y_train_shape = [Y1_train.shape, Y2_train.shape, Y3_train.shape, Y4_train.shape, Y5_train.shape]

for inc, value in enumerate(x_train_shape):
    print("X{}_train.shape: {}".format(inc+1, value))
    print("Y{}_train.shape: {}".format(inc+1, value))
    print("")
```

```
X1_train.shape: (2350, 18)
Y1_train.shape: (2350, 18)

X2_train.shape: (2350, 8)
Y2_train.shape: (2350, 8)

X3_train.shape: (2350, 8)
Y3_train.shape: (2350, 8)

X4_train.shape: (2350, 10)
Y4_train.shape: (2350, 10)

X5_train.shape: (2350, 12)
Y5_train.shape: (2350, 12)
```

*As seen above, a train set of 80% was created for each different DataFrame and the shape of each is the same size.*

*Then, we scale these data using the MinMaxScaler operation to ensure that each dependent variable train sets are in the same range, and we complete the Standardization process by using the fit_transform() method in the MinMaxScaler for Standardization.*

```
# Use min-max scaler and transform each feature accordingly.
# We put each feature value to a certain range (in general (0,1))

scaler1 = MinMaxScaler(feature_range=(0,1))
scaler2 = MinMaxScaler(feature_range=(0,1))
scaler3 = MinMaxScaler(feature_range=(0,1))
scaler4 = MinMaxScaler(feature_range=(0,1))
scaler5 = MinMaxScaler(feature_range=(0,1))
```

```
# We should scale the x_train to make the standardization

scaled_X1train = scaler1.fit_transform(X1_train)
scaled_X2train = scaler2.fit_transform(X2_train)
scaled_X3train = scaler3.fit_transform(X3_train)
scaled_X4train = scaler4.fit_transform(X4_train)
scaled_X5train = scaler5.fit_transform(X5_train)
```

*We initialize linear regression and fit the train data we created, and we get a linear regression model as an output.*

```
# initialize the linear regression model

linearRegression1 = LinearRegression()
linearRegression2 = LinearRegression()
linearRegression3 = LinearRegression()
linearRegression4 = LinearRegression()
linearRegression5 = LinearRegression()
```

```
# Fit the training data to the model (training)

linearRegression1.fit(scaled_X1train, Y1_train)
linearRegression2.fit(scaled_X2train, Y2_train)
linearRegression3.fit(scaled_X3train, Y3_train)
linearRegression4.fit(scaled_X4train, Y4_train)
linearRegression5.fit(scaled_X5train, Y5_train)
```

*In addition, we transform these data using the transform() method by applying MinMaxScaler to the test data in each model, and we reveal a result by predicting these data.*

```
# We should scale each instance of the test set in the same way as we did the training set and just using the transform method

scaled_X1test = scaler1.transform(X1_test)
scaled_X2test = scaler2.transform(X2_test)
scaled_X3test = scaler3.transform(X3_test)
scaled_X4test = scaler4.transform(X4_test)
scaled_X5test = scaler5.transform(X5_test)
```

```
# Predict the values by using all test data

predict1 = linearRegression1.predict(scaled_X1test)
predict2 = linearRegression2.predict(scaled_X2test)
predict3 = linearRegression3.predict(scaled_X3test)
predict4 = linearRegression4.predict(scaled_X4test)
predict5 = linearRegression5.predict(scaled_X5test)
```

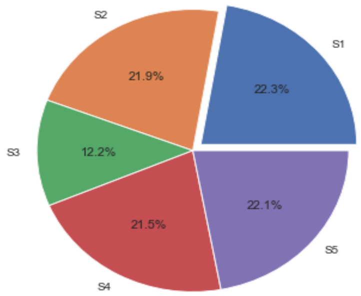
*We look at the Score value and other metric evaluations to evaluate the performances. These metric values are Mean Squared Error and Mean Absolute Error values.*

*In the score value, we want to obtain higher value because the higher score value is closer then the actual value.*

```
# Calculate the score of the model in the test data
# We want to desire higher values

score1 = linearRegression1.score(scaled_X1test, Y1_test)
score2 = linearRegression2.score(scaled_X2test, Y2_test)
score3 = linearRegression3.score(scaled_X3test, Y3_test)
score4 = linearRegression4.score(scaled_X4test, Y4_test)
score5 = linearRegression5.score(scaled_X5test, Y5_test)

score = [score1, score2, score3, score4, score5]
for inc, value in enumerate(score):
    print("score{}: {}".format(inc+1, value))
```

```
score1: 0.8004115433815071
score2: 0.78333006441394
score3: 0.4367334323175648
score4: 0.7717525122198143
score5: 0.7924299173922996
```

```
# Visualizing the obtained score values in a pie chart
Score_Value = [score1, score2, score3, score4, score5]
Score_Label = ['S1', 'S2', 'S3', 'S4', 'S5']

plt.axis('equal')
plt.pie(Score_Value, labels=Score_Label, radius=1.5, autopct='%0.1f%%', explode=[0.1,0,0,0,0])
plt.show()
```

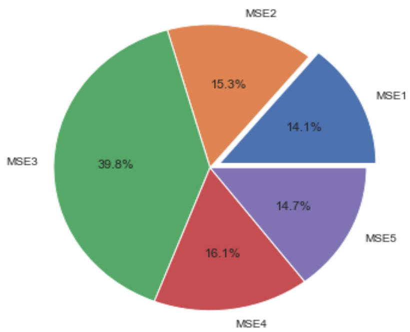



*In the Mean Squared Errror, we want to get lower mean squared error because this is the error value, therefore it should be the lowest number.*

```
# Calculate the mean squared error of the predicted values.
# We want to get lower values

MSE1 = mean_squared_error(Y1_test, predict1)
MSE2 = mean_squared_error(Y2_test, predict2)
MSE3 = mean_squared_error(Y3_test, predict3)
MSE4 = mean_squared_error(Y4_test, predict4)
MSE5 = mean_squared_error(Y5_test, predict5)

MSE = [MSE1, MSE2, MSE3, MSE4, MSE5]
for inc, value in enumerate(MSE):
    print("MSE{}: {}".format(inc+1, value))
```

```
MSE1: 18.7934490307392
MSE2: 20.401858203220733
MSE3: 53.037744315511866
MSE4: 21.492012116665354
MSE5: 19.545006930188862
```

```
# Visualizing the obtained Mean Squared Error values in a pie chart
MSE_Value = [MSE1, MSE2, MSE3, MSE4, MSE5]
MSE_Label = ['MSE1', 'MSE2', 'MSE3', 'MSE4', 'MSE5']

plt.axis('equal')
plt.pie(MSE_Value, labels=MSE_Label, radius=1.5, autopct='%0.1f%%', explode=[0.1,0,0,0,0])
plt.show()
```

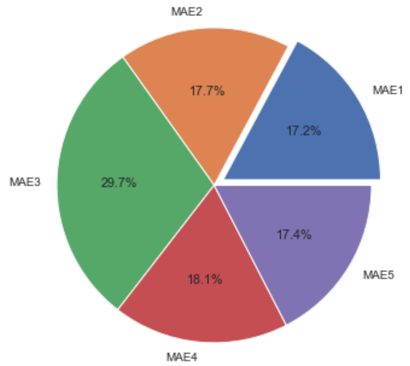


*In the Mean Absolute  Errror, we want to get lower mean absolute error because this is the error value, therefore it should be the lowest number.*

```
# Calculate the Mean Absolute Error of the predicted values

MAE1 = mean_absolute_error(Y1_test, predict1)
MAE2 = mean_absolute_error(Y2_test, predict2)
MAE3 = mean_absolute_error(Y3_test, predict3)
MAE4 = mean_absolute_error(Y4_test, predict4)
MAE5 = mean_absolute_error(Y5_test, predict5)

MAE = [MAE1, MAE2, MAE3, MAE4, MAE5]
for inc, value in enumerate(MAE):
    print("MAE{}: {}".format(inc+1, value))
```

```
MAE1: 3.2487619245102417
MAE2: 3.3427191949184047
MAE3: 5.60722306595964
MAE4: 3.417595579207909
MAE5: 3.2933439234633113
```

```
# Visualizing the obtained Mean Absolute Error values in a pie chart
MAE_Value = [MAE1, MAE2, MAE3, MAE4, MAE5]
MAE_Label = ['MAE1', 'MAE2', 'MAE3', 'MAE4', 'MAE5']

plt.axis('equal')
plt.pie(MAE_Value, labels=MAE_Label, radius=1.5, autopct='%0.1f%%', explode=[0.1,0,0,0,0])
plt.show()
```




## 4 Results

*As can be seen at the end of the Implementation Details section, we look at the Score values and other metric evaluations in the performance evaluation of our Linear regression models. We want it to be the highest value from the 5 different score data we obtained, because the higher the score value, the more similar the predicted values in our linear regression model to the actual value. Thus, we suppress each score value and we get a result like below;*

1. ***score1: 0.8004115433815071***
2. ***score2: 0.78333006441394***
3. ***score3: 0.4367334323175648***
4. ***score4: 0.7717525122198143***
5. ***score5: 0.7924299173922996***

*As seen in the implementation part, Score 1 has the highest value with 22.3%. Also, if we look at the Mean Squance Error result from other metrics, the LinearRegression1 model brings the lowest value with 14.1%. This is the lowest value from the other models value and we want to obtain lowest value for mean squance error value.*

1. ***MSE1: 18.7934490307392***
2. ***MSE2: 20.401858203220733***
3. ***MSE3: 53.037744315511866***
4. ***MSE4: 21.492012116665354***
5. ***MSE5: 19.545006930188862***

*When we look at the Mean Absolute Error values, we see that the lowest error is in the linearRegration1 model with 17.2%.*

1. ***MAE1: 3.2487619245102417***
2. ***MAE2: 3.3427191949184047***
3. ***MAE3: 5.60722306595964***
4. ***MAE4: 3.417595579207909***
5. ***MAE5: 3.2933439234633113***

*According to the performance analysis results, the linearRegrassion1 model has the best performance.*

*LinearRegrassion1;*

1. ***score1: 0.8004115433815071***
2. ***MSE1: 18.7934490307392***
3. ***MAE1: 3.2487619245102417***

*LinearRegrassion2;*

1. ***score2: 0.78333006441394***
2. ***MSE2: 20.401858203220733***
3. ***MAE2: 3.3427191949184047***

*LinearRegrassion3;*

1. ***score3: 0.43673343231756548***
2. ***MSE3: 53.037744315511866***
3. ***MAE3: 5.60722306595964***

*LinearRegrassion4;*

1. ***score4: 0.7717525122198143***
2. ***MSE4: 21.492012116665354***
3. ***MAE4: 3.417595579207909***

*LinearRegrassion5;*

1. ***score5: 0.7924299173922996***
2. ***MSE5: 19.545006930188862***
3. ***MAE5: 3.2933439234633113***

*The best ranking according to Performance Analysis is as follows*
***LinearRegrassion1 > LinearRegrassion5 > LinearRegrassion2 > LinearRegrassion4 > LinearRegrassion3***

*The general formula for the best linear Regrassion model is as follows*

```
# At the end of the Linear Regression, Linear Regression model 1 is the best model
# y = x1*..+x2*...+ intercept (general formula for linear regression model)

str_ = "y="
for i, m in enumerate(linearRegression1.coef_):
    str_ += "x_{}*{}+".format(i, m)

str_ += str(linearRegression1.intercept_)
print(str_)
```

```
y=x_0*-14.340524076757557+x_1*182.60064602773184+x_2*1.4128887304312627+x_3*1.7803638189019073+x_4*-1.0479912132405629+x_5*3.64
84596657590185+x_6*-185.559924820483+x_7*2.3563603812518106+x_8*1.3165851323644284+x_9*3.9803587830314884+x_10*-23.737467263458
96+x_11*3.825805329808659+x_12*-1.5251801240187046+x_13*-0.1351370332208018+x_14*5.770641016770 3925+x_15*12.840634073581944+x_1
6*116181271009580.23+x_17*116181271009578.7+-116181271009523.6
```

*As I mentioned above, we expected df1 or df5 to give the best performance, and as a result, the data in df1, namely the linearRegrassion1 model, gave the best performance because the correlation between the value we will predict and dependent veriable values in this model was very high, so the linearRegrassion1 model showed the best performance.*

## *5 Conclusion*

*In today's world, the field of Computer Engineering not only develops in its own field, but also develops other fields with its relations with other fields and sectors. One of these sectors is the health sector. The usage of artificial intelligence and machine learning in the healthcare system leads to better diagnosis and treatments. For example, estimation of disease outcomes, estimation of surgical outcomes, or estimation of life expectancy with certain data, as we did in this assignment can be given an example. Therefore, machine learning algorithms or statistical analyzes are used more widely in the healthcare system, helping this sector.*

```
# Sorting the correlations between Dependent & Independent Veriable
df.corr(method="pearson").iloc[0, :].abs().sort_values(ascending = False)
```

```
Life expectancy                  1.000000
Schooling                        0.715066
Adult Mortality                  0.696359
Income composition of resources  0.692483
 BMI                             0.559255
 HIV/AIDS                        0.556457
Status_Developing                0.481962
Status_Developed                 0.481962
Diphtheria                       0.475418
 thinness  1-19 years            0.472162
 thinness 5-9 years              0.466629
Polio                            0.461574
GDP                              0.430493
Alcohol                          0.391598
percentage expenditure           0.381791
under-five deaths                0.222503
Total expenditure                0.207981
Hepatitis B                      0.203771
infant deaths                    0.196535
Name: Life expectancy , dtype: float64
```

*Figure 2 – The Correlation between Independent Variable & Dependent Variable*

*As seen in this assignment, the life expectancy of countries was predicted by creating a Linear Regression model with some health data given. As seen in Figure 2, it is observed that Adult Mortality data has a high effect on life expectancy. In addition, it seems to have an important effect on the determination of life expectancy in diseases such as HIV/AIDS and Diphtheria.*

*As a result, Machine Learning & Artificial Intelligence, which are accepted as the specialization area of Computer Science, seem to make a positive contribution to the Health sector and other sector as well.*