Life Expectancy by Linear Regression Report (Assignment 1)

Lecture: CS452/552 – Data Science with Python

University: Ozyegin University

Assignment Prepared by: Furkan Kınlı

Instructor of the Lecture: Assistant Prof. Reyhan Aydoğan

Tuna Tuncer – S018474

1. Introduction

What is the project?

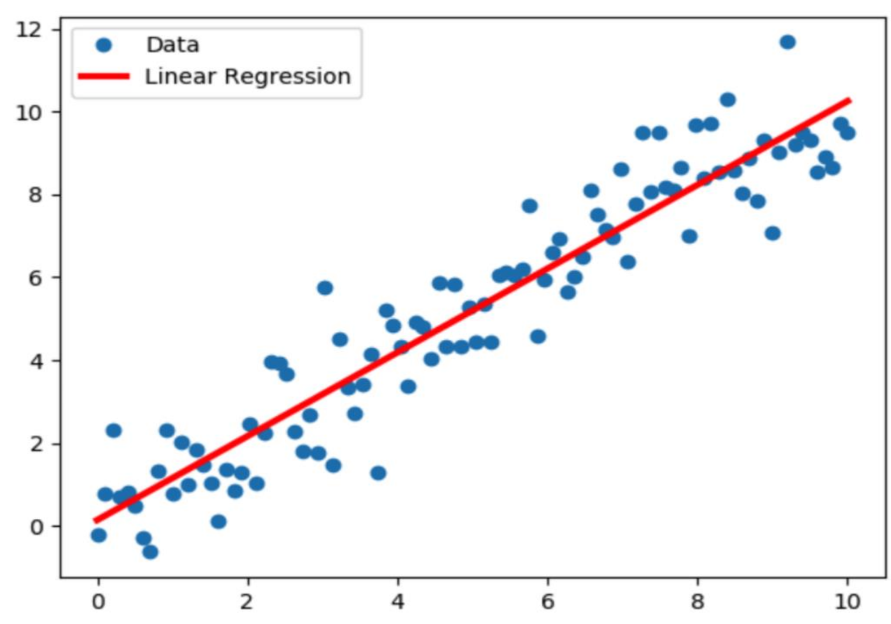
This project is prepared to create predictions on various datasets using data science and machine learning techniques.

Aim of the project

In this project, I was expected to analyze various health-related data from different countries and to develop a linear regression model based on this data to predict *life expectancy* as accurately as possible.

Linear Regression on a Nutshell

Linear regression can be considered as one of the most basic algorithms used in machine learning and statistics. Linear regression assumes that independent features in our data follow a *linear relationship* to form the dependent feature. If there is only one independent feature in the data, the linear regression model used is known as ***simple linear regression***; if there are more than one independent feature, the linear regression model used is referred to as ***multiple linear regression***. I have used multiple linear regression in the project, but for easier understanding, the photo below is about simple linear regression.



[1] Figure 1: Simple Linear Regression model sample illustration

As seen in Figure 1, finding the linear function that best fits the 2-dimensional data is called linear regression.

General Information About Data

The dataset comprises 22 different health-related features from 193 different countries between the years 2000-2015. It has 2938 entries.

Exploring the Features

1- *Country*: (categorical) (e.g., Afghanistan, Zimbabwe, …)

2- *Year*: (numerical) (e.g., 2000, 2015, …)

3- *Status*: Country's status of development (categorical) (e.g., Developing, Developed)

4- *Life expectancy*: (numeric) (e.g., 65, 45.3)

5- *Adult mortality*: [2] “probability of dying between 15 and 60 years per 1000 population” (numeric) (e.g., 263, 665)

6- *Infant death*: (numeric) (e.g., 62, 24, …)

7- *Alcohol*: [3] “in liters of pure alcohol per capita” (numeric) (e.g., 0.01, 6.09, …)

8- *Percentage expenditure*: (numeric) (e.g., 181.9083783, 43.59522964, …)

9- *Hepatitis B*: (numeric) (e.g., 9, 65, …)

10- *Measles*: number of measles cases (numeric) (e.g., 35, 1154, …)

11- *BMI*: (numeric) (e.g., 19.1, 25.5, …)

12- *Under Five Deaths*: (numeric) (e.g., 83, 39, …)

13- *Polio*: (numeric) (e.g., 6, 78, …)

14- *Total expenditure*: (numeric) (e.g., 5.37, 8.16, …)

15- *Diphtheria*: (numeric) (e.g., 65, 85, …)

16- *HIV/AIDS*: (numeric) (e.g., 0.1, 20.5, …)

17- *GDP*: (numeric) (e.g., 584.25921, 53.2772217, …)

18- *Population*: (numeric) (e.g., 33736494, 13124267, …)

19- *Thinness 1-19 years*: (numeric) (e.g., 8.6, 17.2, …)

20- *Thinness 5-9 years*: (numeric) (e.g., 11.2, 17.3, …)

21- *Income composition of resources*: (numeric) (e.g., 0.497, 0.7, …)

22- *Schooling*: (numeric) (e.g., 9.3, 15.5, …)

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	Year	Status	Life expec	Adult Mortality	infant dea	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Populatio	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling
2	2015	Developing	65	263	62	0.01	71.27962362	65	1154	19.1	83	6	8.16	65	0.1	584.2592	33736494	17.2	17.3	0.479	10.1
3	2014	Developing	59.9	271	64	0.01	73.52358168	62	492	18.6	86	58	8.18	62	0.1	612.6965	327582	17.5	17.5	0.476	10
4	2013	Developing	59.9	268	66	0.01	73.21924272	64	430	18.1	89	62	8.13	64	0.1	631.745	31731688	17.7	17.7	0.47	9.9
5	2012	Developing	59.5	272	69	0.01	78.1842153	67	2787	17.6	93	67	8.52	67	0.1	669.959	3696958	17.9	18	0.463	9.8
6	2011	Developing	59.2	275	71	0.01	7.097108703	68	3013	17.2	97	68	7.87	68	0.1	63.53723	2978599	18.2	18.2	0.454	9.5
7	2010	Developing	58.8	279	74	0.01	79.67936736	66	1989	16.7	102	66	9.2	66	0.1	553.3289	2883167	18.4	18.4	0.448	9.2
8	2009	Developing	58.6	281	77	0.01	56.76221682	63	2861	16.2	106	63	9.42	63	0.1	445.8933	284331	18.6	18.7	0.434	8.9
9	2008	Developing	58.1	287	80	0.03	25.87392536	64	1599	15.7	110	64	8.33	64	0.1	373.3611	2729431	18.8	18.9	0.433	8.7
10	2007	Developing	57.5	295	82	0.02	10.91015598	63	1141	15.2	113	63	6.73	63	0.1	369.8358	26616792	19	19.1	0.415	8.4
11	2006	Developing	57.3	295	84	0.03	17.17151751	64	1990	14.7	116	58	7.43	58	0.1	272.5638	2589345	19.2	19.3	0.405	8.1
12	2005	Developing	57.3	291	85	0.02	1.388647732	66	1296	14.2	118	58	8.7	58	0.1	25.29413	257798	19.3	19.5	0.396	7.9
13	2004	Developing	57	293	87	0.02	15.29606643	67	466	13.8	120	5	8.79	5	0.1	219.1414	24118979	19.5	19.7	0.381	6.8
14	2003	Developing	56.7	295	87	0.01	11.08905273	65	798	13.4	122	41	8.82	41	0.1	198.7285	2364851	19.7	19.9	0.373	6.5
15	2002	Developing	56.2	3	88	0.01	16.88735091	64	2486	13	122	36	7.76	36	0.1	187.846	21979923	19.9	2.2	0.341	6.2
16	2001	Developing	55.3	316	88	0.01	10.5747282	63	8762	12.6	122	35	7.8	33	0.1	117.497	2966463	2.1	2.4	0.34	5.9
17	2000	Developing	54.8	321	88	0.01	10.42496	62	6532	12.2	122	24	8.2	24	0.1	114.56	293756	2.3	2.5	0.338	5.5
18	2015	Developing	77.8	74	0	4.6	364.9752287	99	0	58	0	99	6	99	0.1	3954.228	28873	1.2	1.3	0.762	14.2
19	2014	Developing	77.5	8	0	4.51	428.7490668	98	0	57.2	1	98	5.88	98	0.1	4575.764	288914	1.2	1.3	0.761	14.2
20	2013	Developing	77.2	84	0	4.76	430.8769785	99	0	56.5	1	99	5.66	99	0.1	4414.723	289592	1.3	1.4	0.759	14.2
21	2012	Developing	76.9	86	0	5.14	412.4433563	99	9	55.8	1	99	5.59	99	0.1	4247.614	2941	1.3	1.4	0.752	14.2
22	2011	Developing	76.6	88	0	5.37	437.0621	99	28	55.1	1	99	5.71	99	0.1	4437.179	295195	1.4	1.5	0.738	13.3
23	2010	Developing	76.2	91	1	5.28	41.82275719	99	10	54.3	1	99	5.34	99	0.1	494.3588	291321	1.4	1.5	0.725	12.5
24	2009	Developing	76.1	91	1	5.79	348.0559517	98	0	53.5	1	98	5.79	98	0.1	4114.137	2927519	1.5	1.6	0.721	12.2
25	2008	Developing	75.3	1	1	5.61	36.62206845	99	0	52.6	1	99	5.87	99	0.1	437.5396	2947314	1.6	1.6	0.713	12
26	2007	Developing	75.9	9	1	5.58	32.24655228	98	22	51.7	1	99	6.1	98	0.1	363.1369	29717	1.6	1.7	0.703	11.6
27	2006	Developing	74.2	99	1	5.31	3.3021542	98	68	5.8	1	97	5.86	97	0.1	35.1293	2992547	1.7	1.8	0.696	11.4
28	2005	Developing	73.5	15	1	5.16	26.99312143	98	6	49.9	1	97	6.12	98	0.1	279.1429	311487	1.8	1.8	0.685	10.8
29	2004	Developing	73	17	1	4.54	221.8428	99	7	48.9	1	98	6.38	97	0.1	2416.588	326939	1.8	1.9	0.681	10.9
30	2003	Developing	72.8	18	1	4.29	14.71928882	97	8	47.9	1	97	6.27	97	0.1	189.6816	339616	1.9	2	0.674	10.7
31	2002	Developing	73.3	15	1	3.73	104.5169157	96	16	46.9	1	98	6.3	98	0.1	1453.643	3511	2	2.1	0.67	10.7
32	2001	Developing	73.6	14	1	4.25	96.20557078	96	18	46	1	97	6	97	0.1	1326.973	36173	2.1	2.1	0.662	10.6

My Expectations

When I first examined the dataset, I believed that the correlation of schooling, BMI, total expenditure, and percentage expenditure values with life expectancy would be quite high. Because if values such as schooling, total expenditure and percentage expenditure are high, this country is aware of the importance given to health and has the resources to improve health services. I also believed that BMI would have a very high impact as it is a metric that shows people's health very closely.

On the other hand, I predicted that year, country, and population values would have a very low correlation with life expectancy, so I thought I would probably drop these features.

2. Methodology

I will start the project by doing research about the data first. I will start with questions such as which column means what, what can these columns add to the model. Then I will do statistical research on the data and check if there are any null values in the data. Then I will measure the correlation of each feature with the target feature. I will decide which features to drop or impute according to whether there is a null value in the data and the correlation of these features with the target feature. If I have categorical features, I will convert these features to numeric features with one-hot encoding technique. Next, I will scale and split my data and fit my linear regression model. Finally, I will measure my model's score and error rates.

3. Implementation Details

Python was utilized as the coding language in the project, while *Jupyter Notebook* was used as the editor. In addition, P*andas* was used to load the data and preprocessing it, *NumPy* to extract the data to be used in visualization, *Matplotlib* for all visualization purposes, *scikit-learn* to split and scale the data, fit the linear regression model, and calculate the score and error rates of the model.

```python
# I will be using the following libraries to load the data, handle the dataframe, visualize, split the data,
# scale the data, run the linear regression model and calculate the scores of the model.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Loading and Analyzing the Data

Since the data is in csv (comma separated values) form, I load the data using the "read_csv()" method in the pandas library and saved it in the *dataframe* named "df" in Figure 4.

```python
# Loading the csv data into a dataframe
df = pd.read_csv("assignment-1-data.csv")
```

Figure 4

In Figure 5, I printed the first 5 rows to make sure the data was loaded correctly and saw that it loaded without errors as I expected.

```
In [3]:  # Let's see the first 5 entries of the data.
         df.head()
```

Out[3]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1	584.
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1	612.
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1	631.
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1	669.
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1	63.

5 rows × 22 columns

Figure 5

I ran the code in Figure 6 to get an overview of the columns. From the output it can be seen that the dataframe named df has 2938 rows and 22 columns as I expected. As seen in the *Non-null count* section, there are null values in some columns, how I deal with them can be seen in the rest of the report.

```
# Getting the general information about columns.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Country                          2938 non-null   object 
 1   Year                             2938 non-null   int64  
 2   Status                           2938 non-null   object 
 3   Life expectancy                  2928 non-null   float64
 4   Adult Mortality                  2928 non-null   float64
 5   infant deaths                    2938 non-null   int64  
 6   Alcohol                          2744 non-null   float64
 7   percentage expenditure           2938 non-null   float64
 8   Hepatitis B                      2385 non-null   float64
 9   Measles                          2938 non-null   int64  
 10   BMI                             2904 non-null   float64
 11  under-five deaths                2938 non-null   int64  
 12  Polio                            2919 non-null   float64
 13  Total expenditure                2712 non-null   float64
 14  Diphtheria                       2919 non-null   float64
 15   HIV/AIDS                        2938 non-null   float64
 16  GDP                              2490 non-null   float64
 17  Population                       2286 non-null   float64
 18   thinness  1-19 years            2904 non-null   float64
 19   thinness 5-9 years              2904 non-null   float64
 20  Income composition of resources  2771 non-null   float64
 21  Schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

Figure 6

Before fitting the model, any null cells must be eliminated. I saw in Figure 6 that the data I used had null values. As a result, in Figure 7, I printed the number of null values in each column. It can be seen from the output that the 3 columns with the most missing values are population, hepatitis b, and GDP, respectively.

```
In [6]: # Checking all the columns if there are any null valued cells.
        df.isnull().sum()

Out[6]: 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
```

Figure 7

In Figure 8, I suppressed the *Pearson correlations* between all the features in order to understand which features are correlated so that I can get rid of the features that will enter the model if they have little or no effect on life expectancy. I saw that the correlations of population, infant death, measles, and year features with life expectancy were less than 0.2. So, the effects were minimal. So, I will get rid of these columns in the data preprocessing section of the report.

In [8]:

```python
# Getting the pearson correlation between all of the features.
df.corr(method="pearson")
```

Out[8]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphth
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	0.134
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	0.479
Adult Mortality	-0.079052	-0.696359	1.000000	0.078756	-0.195848	-0.242860	-0.162476	0.031176	-0.387017	0.094146	-0.274823	-0.115281	-0.275
infant deaths	-0.037415	-0.196557	0.078756	1.000000	-0.115638	-0.085612	-0.223566	0.501128	-0.227279	0.996629	-0.170689	-0.128616	-0.175
Alcohol	-0.052990	0.404877	-0.195848	-0.115638	1.000000	0.341285	0.087549	-0.051827	0.330408	-0.112370	0.221734	0.296942	0.222
percentage expenditure	0.031400	0.381864	-0.242860	-0.085612	0.341285	1.000000	0.016274	-0.056596	0.228700	-0.087852	0.147259	0.174420	0.143
Hepatitis B	0.104333	0.256762	-0.162476	-0.223566	0.087549	0.016274	1.000000	-0.120529	0.150380	-0.233126	0.486171	0.058280	0.611
Measles	-0.082493	-0.157586	0.031176	0.501128	-0.051827	-0.056596	-0.120529	1.000000	-0.175977	0.507809	-0.136166	-0.106241	-0.141
BMI	0.108974	0.567694	-0.387017	-0.227279	0.330408	0.228700	0.150380	-0.175977	1.000000	-0.237669	0.284569	0.242503	0.283
under-five deaths	-0.042937	-0.222529	0.094146	0.996629	-0.112370	-0.087852	-0.233126	0.507809	-0.237669	1.000000	-0.188720	-0.130148	-0.195
Polio	0.094158	0.465556	-0.274823	-0.170689	0.221734	0.147259	0.486171	-0.136166	0.284569	-0.188720	1.000000	0.137330	0.673
Total expenditure	0.090740	0.218086	-0.115281	-0.128616	0.296942	0.174420	0.058280	-0.106241	0.242503	-0.130148	0.137330	1.000000	0.152
Diphtheria	0.134337	0.479495	-0.275131	-0.175171	0.222020	0.143624	0.611495	-0.141882	0.283147	-0.195668	0.673553	0.152754	1.000
HIV/AIDS	-0.139741	-0.556556	0.523821	0.025231	-0.048845	-0.097857	-0.112675	0.030899	-0.243717	0.038062	-0.159560	-0.001389	-0.164
GDP	0.101620	0.461455	-0.296049	-0.108427	0.354712	0.899373	0.083903	-0.076466	0.301557	-0.112081	0.211976	0.138364	0.200
Population	0.016969	-0.021538	-0.013647	0.556801	-0.035252	-0.025662	-0.123321	0.265966	-0.072301	0.544423	-0.038540	-0.079662	-0.028
thinness 1-19 years	-0.047876	-0.477183	0.302904	0.465711	-0.428795	-0.251369	-0.120429	0.224808	-0.532025	0.467789	-0.221823	-0.277101	-0.229
thinness 5-9 years	-0.050929	-0.471584	0.308457	0.471350	-0.417414	-0.252905	-0.124960	0.221072	-0.538911	0.472263	-0.222592	-0.283774	-0.222
Income composition of resources	0.243468	0.724776	-0.457626	-0.145139	0.450040	0.381952	0.199549	-0.129568	0.508774	-0.163305	0.381078	0.166682	0.401
Schooling	0.209400	0.751975	-0.454612	-0.193720	0.547378	0.389687	0.231117	-0.137225	0.546961	-0.209373	0.417866	0.246384	0.425

Figure 8

Next, I decided to draw a graph where the correlation of all features with life expectancy can be seen, and colored the columns with correlation values between -0.2 and 0.2 in red. As can be seen in Figure 9, the correlation of year, infant deaths, measles, and population columns is quite low.

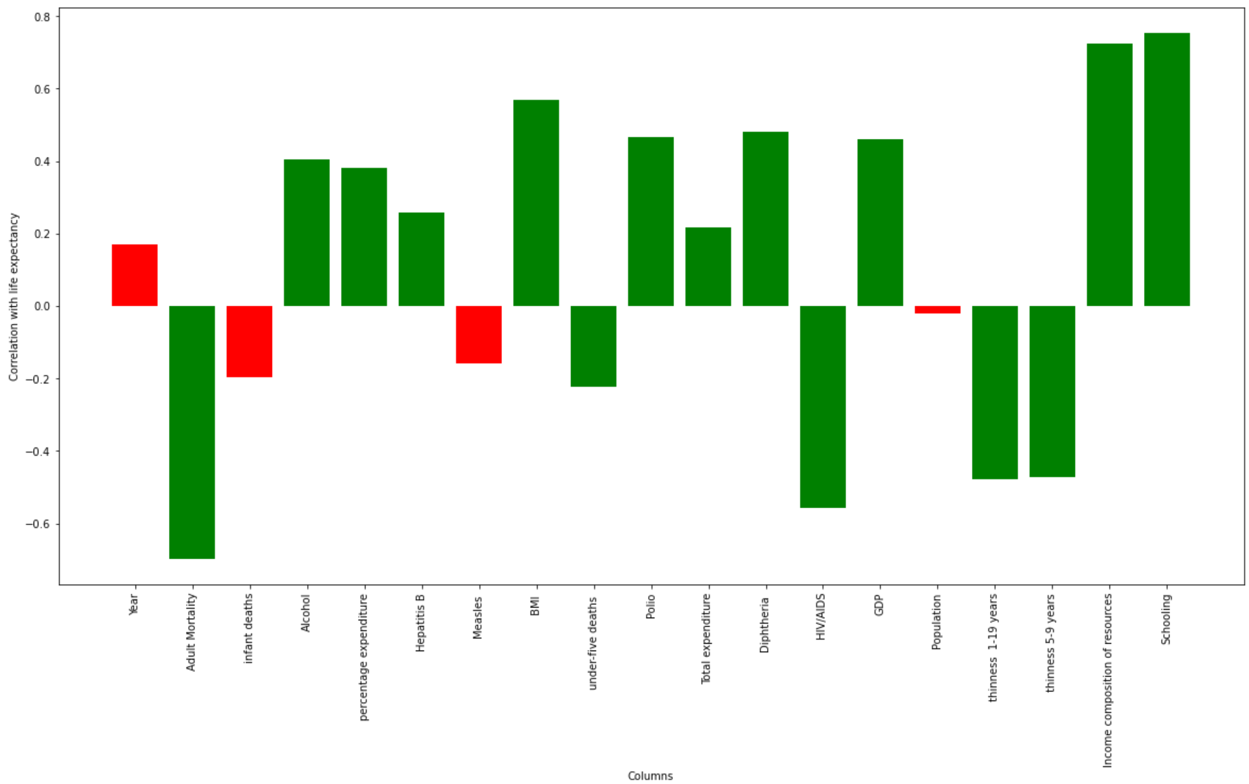


Figure 9

As you can see in Figure 10, I drew a histogram for each feature to see the distribution of numeric features, so that information such as in which value ranges the feature is in and at which values it is concentrated can be accessed visually.

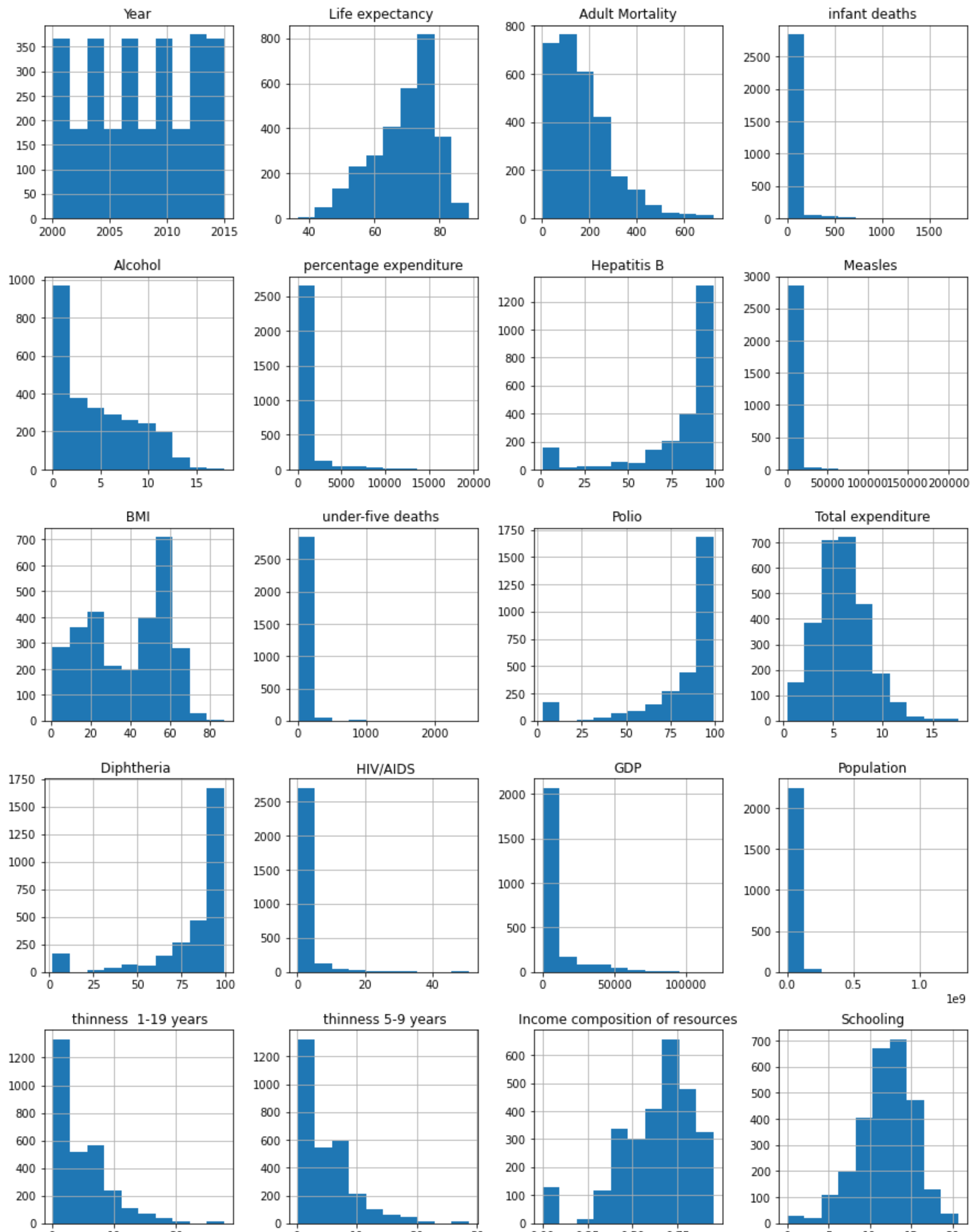

Year
Life expectancy
Adult Mortality
infant deaths
Alcohol
percentage expenditure
Hepatitis B
Measles
BMI
under-five deaths
Polio
Total expenditure
Diphtheria
HIV/AIDS
GDP
Population
1e9
thinness 1-19 years
thinness 5-9 years
Income composition of resources
Schooling

Figure 10

Preprocessing the Data

In this part, I used 3 different techniques called dropping, imputing and one-hot encoding to process the data.

- *<u>Dropping</u>*

In Figure 11, since I don't know if there is any duplication in the data, I got rid of them and checked the shape of the dataframe and saw that it did not change. It means that there is no duplicate row in the data.

```
In [14]:  # Let's remove the duplicated columns (if there are any) just in case.
          df.drop_duplicates(inplace=True)

          df.shape

Out[14]:  (2938, 22)
```

Figure 11

In Figure 12, in addition to the 4 features (population, infant death, meatles and year), which we had previously decided to have little effect by looking at their correlations, I thought that the "country" feature would not contribute to the model (ie, it did not affect life expectancy). I dropped them and checked if they were successfully dropped.

I also noticed that there are 10 null values in the life expectancy column. I didn't want to impute null values because that's the value we're going to predict anyway.

```
# Removing some of the columns from the dataframe.
df.drop(['Country', 'Year', 'infant deaths', 'Measles ', 'Population'], axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2938 entries, 0 to 2937
Data columns (total 17 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Status                           2938 non-null   object 
 1   Life expectancy                  2928 non-null   float64
 2   Adult Mortality                  2928 non-null   float64
 3   Alcohol                          2744 non-null   float64
 4   percentage expenditure           2938 non-null   float64
 5   Hepatitis B                      2385 non-null   float64
 6    BMI                             2904 non-null   float64
 7   under-five deaths                2938 non-null   int64  
 8   Polio                            2919 non-null   float64
 9   Total expenditure                2712 non-null   float64
 10  Diphtheria                       2919 non-null   float64
 11   HIV/AIDS                        2938 non-null   float64
 12  GDP                              2490 non-null   float64
 13   thinness  1-19 years            2904 non-null   float64
 14   thinness 5-9 years              2904 non-null   float64
 15  Income composition of resources  2771 non-null   float64
 16  Schooling                        2775 non-null   float64
dtypes: float64(15), int64(1), object(1)
memory usage: 413.2+ KB
```

Figure 12

So, I dropped these 10 rows in figure 13 and checked if they were dropped.

```
In [16]: # Removing the null valued Life expectancy rows.
         df = df[df['Life expectancy '].notna()]

In [17]: # Checking if the number of rows are decreased by 10.
         df.shape

Out[17]: (2928, 17)
```

Figure 13

- *Imputation*

In Figure 14, since I will impute the remaining null values, I checked which columns have null values.

In [18]:
```
# Checking all the columns if there are any null valued cells.
df.isnull().sum().sort_values(ascending=False)
```

Out[18]:
```
Hepatitis B                        553
GDP                                443
Total expenditure                  226
Alcohol                            193
Schooling                          160
Income composition of resources    160
 thinness 5-9 years                 32
 BMI                                32
 thinness  1-19 years               32
Diphtheria                          19
Polio                               19
 HIV/AIDS                            0
Life expectancy                      0
under-five deaths                    0
percentage expenditure               0
Adult Mortality                      0
Status                               0
dtype: int64
```

Figure 14

In Figure 15, I impute all remaining columns with null values by taking the mean of the corresponding columns.

```
In [19]:  # Filling all of the null cell's with the mean of the corresponding feature.
          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)

          df.isnull().sum()

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

Figure 15

- *One-Hot Encoding*

In Figure 16, in order for the status feature to be one-hot encoded, its type must be category. So I set its type as category.

```
In [22]: # Changing the type of the Status to "category" in order to prepare it to be one-hot encoded.
df["Status"] = df["Status"].astype("category")
```

```
In [23]: # Checking if changing type as category is reflected.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2928 entries, 0 to 2937
Data columns (total 17 columns):
 #   Column                           Non-Null Count  Dtype   
---  ------                           --------------  -----   
 0   Status                           2928 non-null   category
 1   Life expectancy                  2928 non-null   float64 
 2   Adult Mortality                  2928 non-null   float64 
 3   Alcohol                          2928 non-null   float64 
 4   percentage expenditure           2928 non-null   float64 
 5   Hepatitis B                      2928 non-null   float64 
 6    BMI                             2928 non-null   float64 
 7   under-five deaths                2928 non-null   int64   
 8   Polio                            2928 non-null   float64 
 9   Total expenditure                2928 non-null   float64 
 10  Diphtheria                       2928 non-null   float64 
 11   HIV/AIDS                        2928 non-null   float64 
 12  GDP                              2928 non-null   float64 
 13   thinness  1-19 years            2928 non-null   float64 
 14   thinness 5-9 years              2928 non-null   float64 
 15  Income composition of resources  2928 non-null   float64 
 16  Schooling                        2928 non-null   float64 
dtypes: category(1), float64(15), int64(1)
memory usage: 391.9 KB
```

Figure 16

In Figure 17, I one-hot encoded the status feature with the *get_dummies* method, and observed that there were two features, *Status_Developed* and *Status_Developing*, as the 16th and 17th columns.

```
In [24]: # One-hot encoding the status feature (since it's the only categorical feature left).
         df = pd.get_dummies(df)
         df.head()
```

Out[24]:

	Life expectancy	Adult Mortality	Alcohol	percentage expenditure	Hepatitis B	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	thinness 1-19 years	thinness 5-9 years	Incor compositi resourc
0	65.0	263.0	0.01	71.279624	65.0	19.1	83	6.0	8.16	65.0	0.1	584.259210	17.2	17.3	0.4
1	59.9	271.0	0.01	73.523582	62.0	18.6	86	58.0	8.18	62.0	0.1	612.696514	17.5	17.5	0.4
2	59.9	268.0	0.01	73.219243	64.0	18.1	89	62.0	8.13	64.0	0.1	631.744976	17.7	17.7	0.4
3	59.5	272.0	0.01	78.184215	67.0	17.6	93	67.0	8.52	67.0	0.1	669.959000	17.9	18.0	0.4
4	59.2	275.0	0.01	7.097109	68.0	17.2	97	68.0	7.87	68.0	0.1	63.537231	18.2	18.2	0.4

```
In [25]: # Checking if numerical status columns are created.
         df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2928 entries, 0 to 2937
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Life expectancy                 2928 non-null   float64
 1   Adult Mortality                 2928 non-null   float64
 2   Alcohol                         2928 non-null   float64
 3   percentage expenditure          2928 non-null   float64
 4   Hepatitis B                     2928 non-null   float64
 5    BMI                            2928 non-null   float64
 6   under-five deaths               2928 non-null   int64
 7   Polio                           2928 non-null   float64
 8   Total expenditure               2928 non-null   float64
 9   Diphtheria                      2928 non-null   float64
 10   HIV/AIDS                       2928 non-null   float64
 11  GDP                             2928 non-null   float64
 12   thinness  1-19 years           2928 non-null   float64
 13   thinness 5-9 years             2928 non-null   float64
 14  Income composition of resources 2928 non-null   float64
 15  Schooling                       2928 non-null   float64
 16  Status_Developed                2928 non-null   uint8
 17  Status_Developing               2928 non-null   uint8
dtypes: float64(15), int64(1), uint8(2)
memory usage: 394.6 KB
```

Figure 17

Thus, my data preprocess ended, and I checked the correlation values again in Figure 18.

```
In [26]: # Checking the correlation of every feature with the life expectancy after the data preprocessing.
         df.corr(method="pearson").iloc[:, 0].sort_values(ascending=False)

Out[26]: Life expectancy                     1.000000
         Schooling                           0.718614
         Income composition of resources     0.692621
          BMI                                0.562453
         Status_Developed                    0.482136
         Diphtheria                          0.476442
         Polio                               0.462592
         GDP                                 0.430551
         Alcohol                             0.392420
         percentage expenditure              0.381864
         Total expenditure                   0.209628
         Hepatitis B                         0.204566
         under-five deaths                  -0.222529
          thinness 5-9 years                -0.467231
          thinness  1-19 years              -0.472778
         Status_Developing                  -0.482136
          HIV/AIDS                          -0.556556
         Adult Mortality                    -0.696359
         Name: Life expectancy , dtype: float64
```

Figure 18

Creating Different Data Frames for Linear Regression Models

I created 5 different data frames to train linear regression models.

```
# Creating 5 different dataframes.
df_1 = df[df.corr(method="pearson").iloc[:, 0].abs()[df.corr(method="pearson").iloc[:, 0].abs() > 0.2].index] # |corr| > 0.2
df_2 = df[df.corr(method="pearson").iloc[:, 0].abs()[df.corr(method="pearson").iloc[:, 0].abs() > 0.21].index] # |corr| > 0.2
df_3 = df[df.corr(method="pearson").iloc[:, 0].abs()[df.corr(method="pearson").iloc[:, 0].abs() > 0.4].index] # |corr| > 0.4
df_4 = df[df.corr(method="pearson").iloc[:, 0].abs()[df.corr(method="pearson").iloc[:, 0].abs() > 0.45].index] # |corr| > 0.4
df_5 = df.loc[:, ~df.columns.isin(['Hepatitis B', 'GDP'])] # (not including "Hepatitis B" and "GDP" because they have the mos

for index, data_frame in enumerate([df_1, df_2, df_3, df_4, df_5]):
    print("Shape of df_{}: {}\n".format(index+1, data_frame.shape))
```

```
Shape of df_1: (2928, 18)

Shape of df_2: (2928, 16)

Shape of df_3: (2928, 13)

Shape of df_4: (2928, 12)

Shape of df_5: (2928, 16)
```

Figure 19

df_1 -> All the features that has the absolute correlation with life expectancy greater than 0.2

df_2 -> All the features that has the absolute correlation with life expectancy greater than 0.21 (not including "Hepatitis B" and "Total expenditure")

df_3 -> All the features that has the absolute correlation with life expectancy greater than 0.4

df_4 -> All the features that has the absolute correlation with life expectancy greater than 0.45 (not including "GDP")

df_5 -> All features but “Hepatitis B" and "GDP" because they have the most number of null values

Splitting the Data and Fitting the Models

In Figure 20, I split the data frames into x (independent features, e.g., polio, BMI) and y (dependent feature e.g., life expectancy).

```python
# Splitting x and y part of the dataframes.
x_1 = df_1.iloc[:, 1:].values
y_1 = df_1.iloc[:, 0].values
print("Shape of x_1:", x_1.shape)
print("Shape of y_1:", y_1.shape, "\n")

x_2 = df_2.iloc[:, 1:].values
y_2 = df_2.iloc[:, 0].values
print("Shape of x_2:", x_2.shape)
print("Shape of y_2:", y_2.shape, "\n")

x_3 = df_3.iloc[:, 1:].values
y_3 = df_3.iloc[:, 0].values
print("Shape of x_3:", x_3.shape)
print("Shape of y_3:", y_3.shape, "\n")

x_4 = df_4.iloc[:, 1:].values
y_4 = df_4.iloc[:, 0].values
print("Shape of x_4:", x_4.shape)
print("Shape of y_4:", y_4.shape, "\n")

x_5 = df_5.iloc[:, 1:].values
y_5 = df_5.iloc[:, 0].values
print("Shape of x_5:", x_5.shape)
print("Shape of y_5:", y_5.shape, "\n")
```

```
Shape of x_1: (2928, 17)
Shape of y_1: (2928,)

Shape of x_2: (2928, 15)
Shape of y_2: (2928,)

Shape of x_3: (2928, 12)
Shape of y_3: (2928,)

Shape of x_4: (2928, 11)
Shape of y_4: (2928,)

Shape of x_5: (2928, 15)
Shape of y_5: (2928,)
```

Figure 20

In Figure 21, I split the x and y's that I had previously separated into 80 percent train and 20 percent test data. Because after training my model, I will need test data to measure its performance. Afterwards, I created 5 different scaler instances to scale all my data between 0 and 1 and scaled the data.

```python
# Splitting the dataframes into two parts as test and train.
x_1_train, x_1_test, y_1_train, y_1_test = train_test_split(x_1, y_1, test_size=0.2, random_state=147)
x_2_train, x_2_test, y_2_train, y_2_test = train_test_split(x_2, y_2, test_size=0.2, random_state=147)
x_3_train, x_3_test, y_3_train, y_3_test = train_test_split(x_3, y_3, test_size=0.2, random_state=147)
x_4_train, x_4_test, y_4_train, y_4_test = train_test_split(x_4, y_4, test_size=0.2, random_state=147)
x_5_train, x_5_test, y_5_train, y_5_test = train_test_split(x_5, y_5, test_size=0.2, random_state=147)
```

```python
# Creating 5 different MinMax scalers for scaling the x data between 0 and 1 for each model.
scaler_1=MinMaxScaler(feature_range=(0,1))
scaler_2=MinMaxScaler(feature_range=(0,1))
scaler_3=MinMaxScaler(feature_range=(0,1))
scaler_4=MinMaxScaler(feature_range=(0,1))
scaler_5=MinMaxScaler(feature_range=(0,1))
```

```python
# Scaling the values of all of the independent variables between the range 0 and 1.
scaled_x_1_train=scaler_1.fit_transform(x_1_train)
scaled_x_2_train=scaler_2.fit_transform(x_2_train)
scaled_x_3_train=scaler_3.fit_transform(x_3_train)
scaled_x_4_train=scaler_4.fit_transform(x_4_train)
scaled_x_5_train=scaler_5.fit_transform(x_5_train)
```

Figure 21

In Figure 22, I created an instance for 5 different models and fit these models using the data I prepared.

```python
# Creating the instance of linear regression model for all of the data frames.
model_1=LinearRegression()
model_2=LinearRegression()
model_3=LinearRegression()
model_4=LinearRegression()
model_5=LinearRegression()
```

```python
# Fitting every model.
model_1.fit(scaled_x_1_train,y_1_train)
model_2.fit(scaled_x_2_train,y_2_train)
model_3.fit(scaled_x_3_train,y_3_train)
model_4.fit(scaled_x_4_train,y_4_train)
model_5.fit(scaled_x_5_train,y_5_train)
```

Figure 22

Characteristics of Models

In Figure 23, I observed the slope and intercept values by suppressing the mathematical equations of the models I had fit.

```python
# Printing the equations of all models.
for index,model in enumerate([model_1, model_2, model_3, model_4, model_5]):
    print("Equation of model_{}:".format(index+1))
    _str="y="
    for i,m in enumerate(model.coef_):
        _str+= "(x^{}*{})+".format(i, m)
    _str+=str(model.intercept_)
    print(_str, "\n")
```

```
Equation of model_1:
y=(x^0*-14.439650233424524)+(x^1*0.3212411600987481)+(x^2*0.8476235457724058)+(x^3*-1.544311529800617)+(x^4*3.12343059139868
4)+(x^5*-3.912814744431224)+(x^6*3.441792920227421)+(x^7*1.0497348724097972)+(x^8*4.418806127133217)+(x^9*-24.97538099332439
6)+(x^10*4.091432313112289)+(x^11*-2.4569709723001036)+(x^12*0.8567202643153158)+(x^13*6.503784583835126)+(x^14*14.420979733
247245)+(x^15*0.8033958397724363)+(x^16*-0.8033958397724337)+53.99157986302422

Equation of model_2:
y=(x^0*-14.44048318406197)+(x^1*0.4157855817769023)+(x^2*1.701393408071506)+(x^3*3.205488194164046)+(x^4*-3.335357617909466)
+(x^5*3.252630687644589)+(x^6*3.856226386433968)+(x^7*-24.78949094565128)+(x^8*3.449269846360505)+(x^9*-2.494281326173818)+
(x^10*0.7057417409606325)+(x^11*6.438236571279405)+(x^12*14.587113417704817)+(x^13*0.8210605310337259)+(x^14*-0.821060531033
7321)+53.604770530607006

Equation of model_3:
y=(x^0*-14.37768218325415)+(x^1*3.161820679662293)+(x^2*3.2923866778276634)+(x^3*3.937703942443354)+(x^4*-24.57524333535414
5)+(x^5*4.8815350406596405)+(x^6*-2.9515715114944454)+(x^7*0.27254827333613546)+(x^8*6.309688281067067)+(x^9*14.823413480297
972)+(x^10*0.8659218328031982)+(x^11*-0.8659218328031982)+53.659372676315

Equation of model_4:
y=(x^0*-14.556175510201468)+(x^1*3.15034579229495)+(x^2*3.345298439123938)+(x^3*3.867469238652796)+(x^4*-24.505482870797078)
+(x^5*-2.778547729716976)+(x^6*-0.005630995069790752)+(x^7*6.860929864838935)+(x^8*15.1164302923241)+(x^9*1.086099510672397
4)+(x^10*-1.0860995106723903)+53.65454891622315

Equation of model_5:
y=(x^0*-14.442632176416506)+(x^1*0.2785758322235755)+(x^2*4.795781431449723)+(x^3*3.1926718535380556)+(x^4*-3.38233999771568
74)+(x^5*3.293073955274346)+(x^6*0.8742111799576904)+(x^7*3.8168857419675275)+(x^8*-24.946645473116597)+(x^9*-2.38781067449
35663)+(x^10*0.6868769803269857)+(x^11*6.702418903817508)+(x^12*14.5083186716066)+(x^13*0.8073941118537138)+(x^14*-0.8073941
11853712)+53.31136608513667
```

Figure 23

4. Results

Visualizing Score and Error Values of Models

As can be seen in Figures 24, 25, 26, *model 1* showed the best performance, while *model 4* showed the worst performance.

I came across an interesting result when examining mean absolute error values. When we look at the score and mean squared error values, we observe that model 1 is more successful than other models, while model 5 is the most successful model according to mean absolute error values.

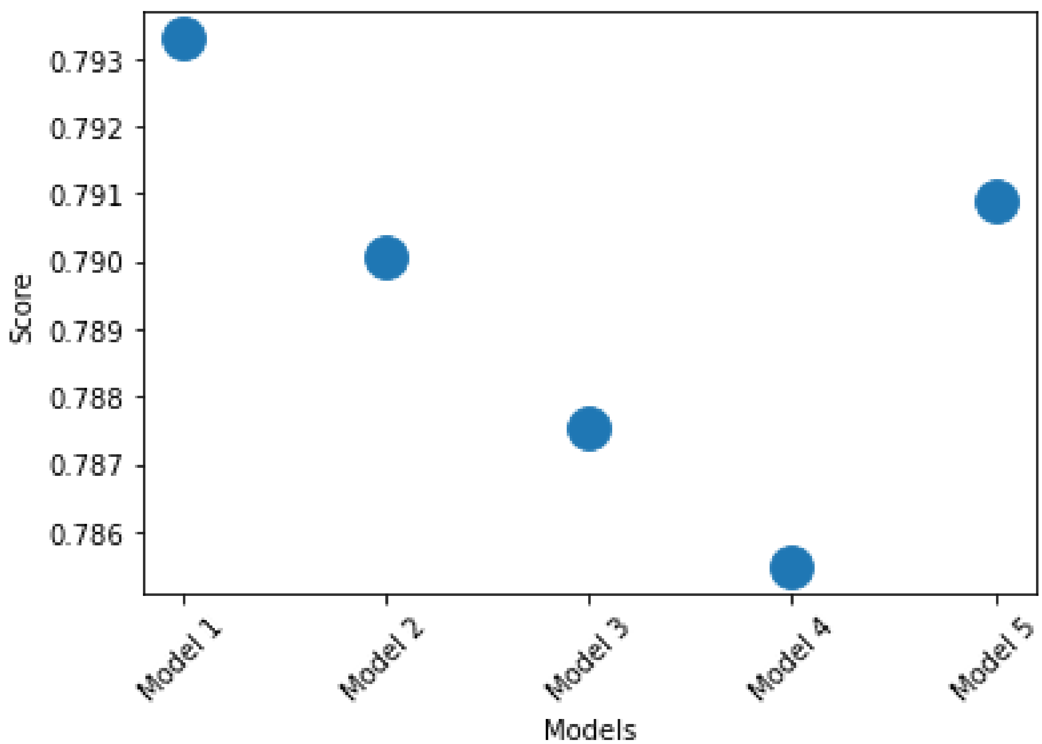


Figure 24

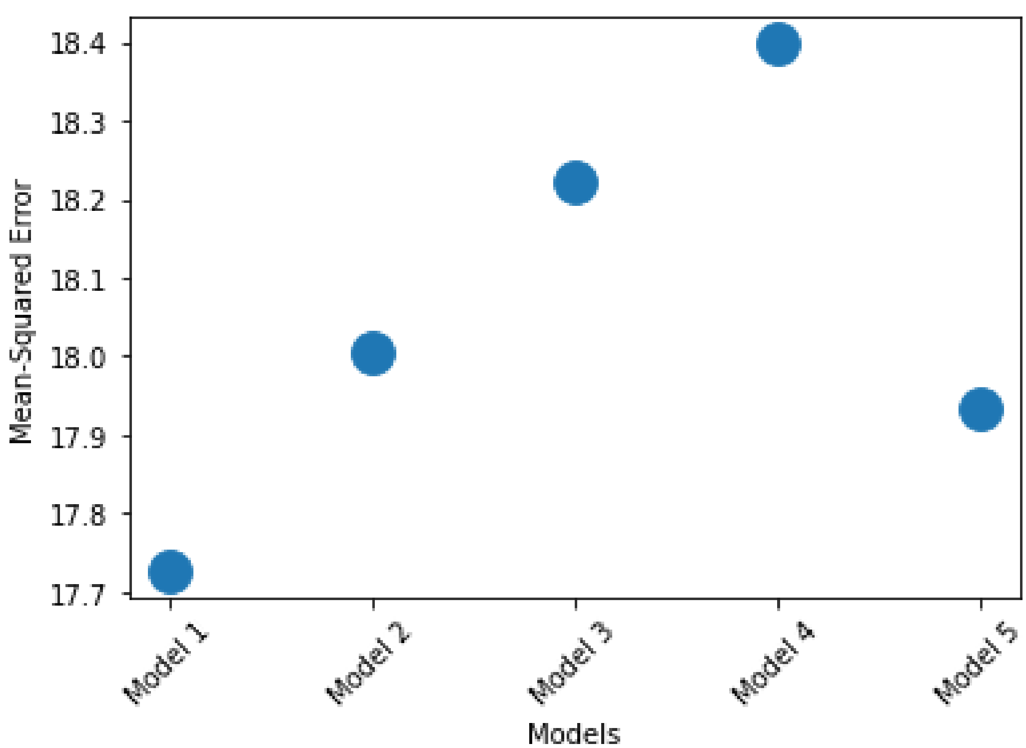


Figure 25

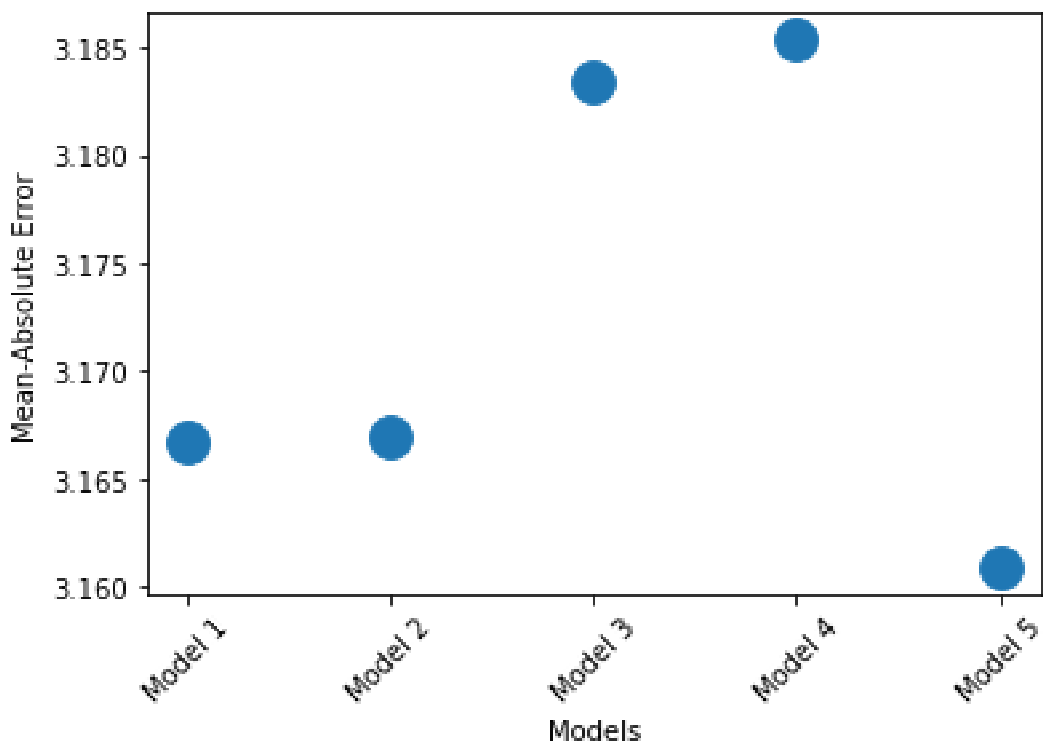


Figure 26

Why First Model was the best?

As seen in the output in Figure 19, *df_1* had the highest number of features (18 columns), while features with low correlation were excluded in other data frames. While getting rid of low correlation features, the size of the data frame is also getting smaller. So, the negative effect of shrinking the size of the data frame is stronger than the positive effect of getting rid of low-correlated features.

5. Conclusion

As a result, within the scope of this project, I processed the data and created 5 different models and measured the score and error rates of each model. The best linear regression model I have produced has an accuracy of 79.32%. One of the most important and also predicted results was the realization that the correlation between schooling value and life expectancy was as high as 0.71, as seen in figure 18. Because in the light of this finding, it has been observed that if the importance given to schooling is increased, the life expectancy of people will also be increased.

[1] Survey of Machine Learning and Data Mining Techniques used in Multimedia System - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Linear-Regression-model-sample-illustration_fig3_333457161 [accessed 14 Nov, 2021]

[2] WHO, 2018, Adult Mortality Rate, https://www.who.int/data/gho/data/indicators/indicator-details/GHO/adult-mortality-rate-(probability-of-dying-between-15-and-60-years-per-1000-population)

[3] World Population Review, 2021, Alcohol Consumption By Country 2021, https://worldpopulationreview.com/country-rankings/alcohol-consumption-by-country