Life Expectancy Data Science Project

Data Set is obtainted from WHO, source here.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

Describing datasets

First step is calculating facts and visualizing our dataset - this will help us determine to what dept we should go with cleaning our data.

In [3]:
```
df = pd.read_csv("./data/LifeExpectancyData.csv", delimiter=',')
df.head(20)
```

Out[3]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	H
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	
5	Afghanistan	2010	Developing	58.8	279.0	74	0.01	79.679367	
6	Afghanistan	2009	Developing	58.6	281.0	77	0.01	56.762217	
7	Afghanistan	2008	Developing	58.1	287.0	80	0.03	25.873925	
8	Afghanistan	2007	Developing	57.5	295.0	82	0.02	10.910156	
9	Afghanistan	2006	Developing	57.3	295.0	84	0.03	17.171518	
10	Afghanistan	2005	Developing	57.3	291.0	85	0.02	1.388648	
11	Afghanistan	2004	Developing	57.0	293.0	87	0.02	15.296066	
12	Afghanistan	2003	Developing	56.7	295.0	87	0.01	11.089053	
13	Afghanistan	2002	Developing	56.2	3.0	88	0.01	16.887351	
14	Afghanistan	2001	Developing	55.3	316.0	88	0.01	10.574728	
15	Afghanistan	2000	Developing	54.8	321.0	88	0.01	10.424960	
16	Albania	2015	Developing	77.8	74.0	0	4.60	364.975229	
17	Albania	2014	Developing	77.5	8.0	0	4.51	428.749067	
18	Albania	2013	Developing	77.2	84.0	0	4.76	430.876979	
19	Albania	2012	Developing	76.9	86.0	0	5.14	412.443356	

20 rows × 22 columns

For more detailed statistics about our dataset, like how many rows in column are unique (for categorical dataset), which values are the most common...For numerical columns, we also get basic statistics calculated like mean, standard deviation, min value, and also quantil values.

In [4]:
```
df.describe(include='all')
```

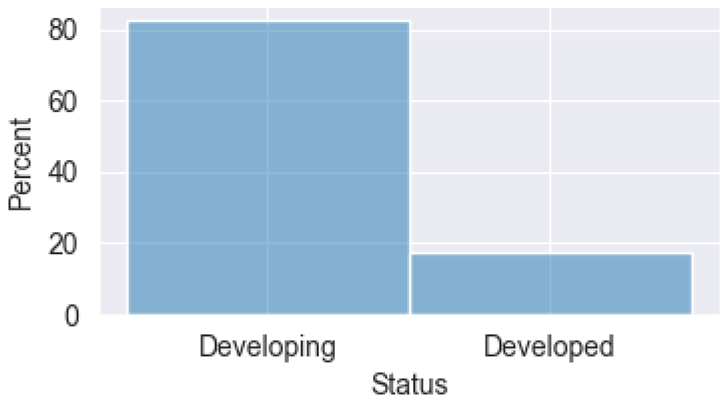
Out[4]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	A
count	2938	2938.000000	2938	2928.000000	2928.000000	2938.000000	2744.
unique	193	NaN	2	NaN	NaN	NaN	
top	Afghanistan	NaN	Developing	NaN	NaN	NaN	
freq	16	NaN	2426	NaN	NaN	NaN	
mean	NaN	2007.518720	NaN	69.224932	164.796448	30.303948	4.
std	NaN	4.613841	NaN	9.523867	124.292079	117.926501	4.
min	NaN	2000.000000	NaN	36.300000	1.000000	0.000000	0.
25%	NaN	2004.000000	NaN	63.100000	74.000000	0.000000	0.
50%	NaN	2008.000000	NaN	72.100000	144.000000	3.000000	3.
75%	NaN	2012.000000	NaN	75.700000	228.000000	22.000000	7.
max	NaN	2015.000000	NaN	89.000000	723.000000	1800.000000	17.

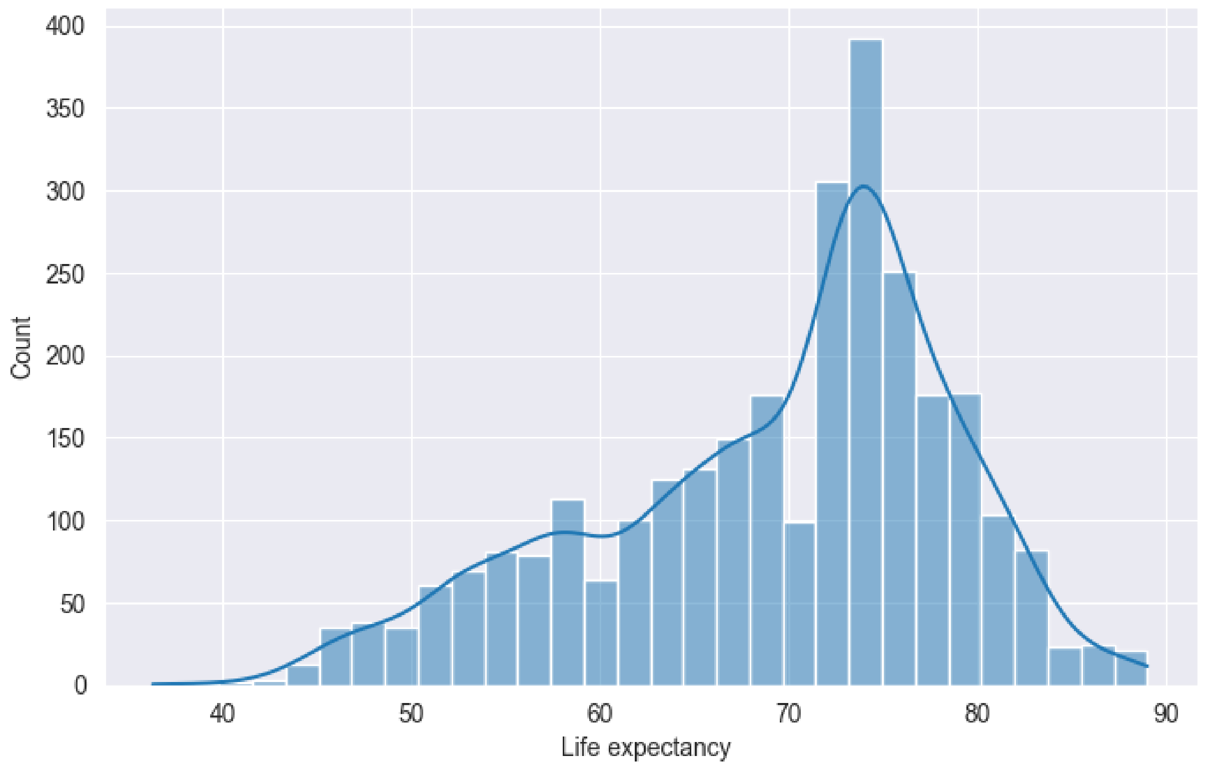
11 rows × 22 columns

As we can see, only two columns are categorical - `Country` and `Status`. Country is pretty much difficult to visualize, as there are 193 values. But we can use histogram for visualization of column `Status`. I choose to visualize it with percent, I don't need to know exact count of each value in this column.

In [5]:
```
plt.figure(figsize=(4,2))
sns.histplot(df.Status,  stat="percent", discrete=True, alpha=0.5)
plt.show()
```

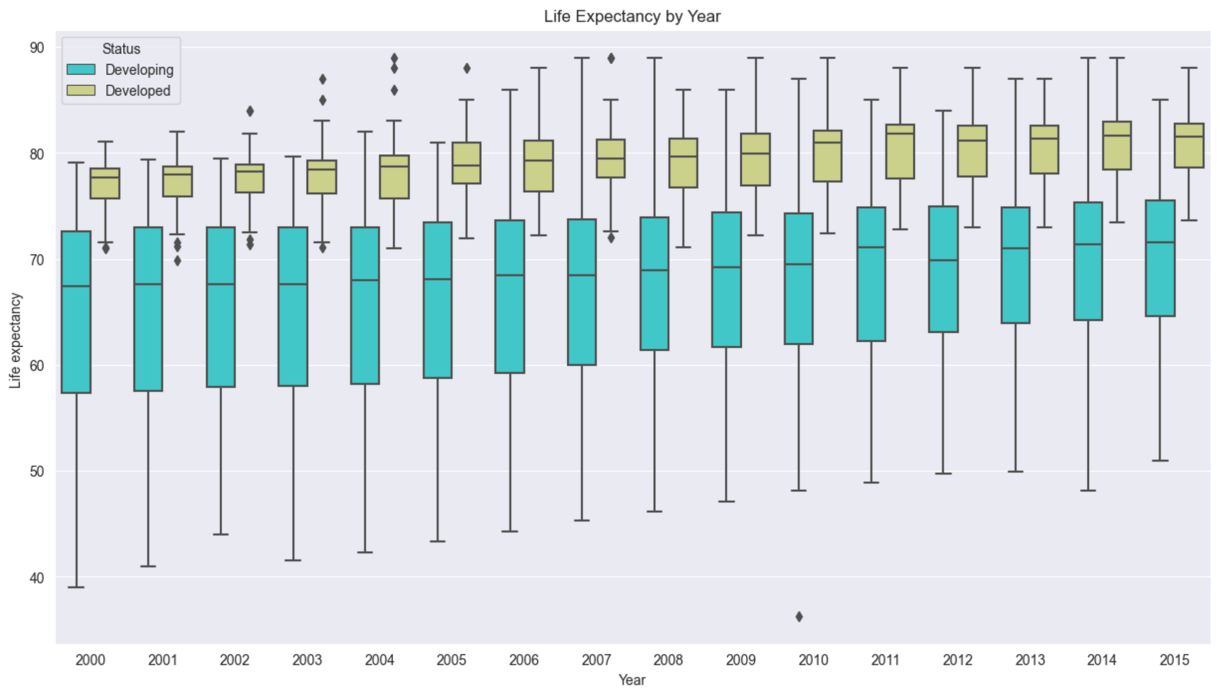


In [6]:
```
plt.figure(figsize=(8,5))
sns.histplot(df['Life expectancy '], bins=30, kde=True)
plt.show()
```



It is possible to create more complex graph, eg. `Box plot` for visualizating some more complex relationships between data. For example, in the next graph, there is visualized life expectancy by year.

```
In [7]: plt.figure(figsize=(15,8))
        plt.title("Life Expectancy by Year")
        sns.boxplot(x='Year',y='Life expectancy ',data=df, palette='rainbow', hue='Status')
        plt.show()
```



Cleaning data

Missing values

There are two strategies to handle missing values:

- delete rows containing missing values - generally not the best solution, mostly when dealing with smaller datasets - we could delete some useful data, or make more unweighted dataset
- fill them with some value - with constant - with mean - with mode - with median - with previous/next value - with most frequent value (categorical data) - with some new default value, eg. `Missing` (categorical data) - using more complex techniques for filling (eg. if there is some correlation between columns, we could use kNN for computing them)

In [8]:
```
# getting number of null values in each column
df.isnull().sum()
```

Out[8]:
```
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
```

In [9]:
```
# Filling missing values
imputer = SimpleImputer()
df['Life expectancy '] = imputer.fit_transform(df[['Life expectancy ']])
df['Adult Mortality'] = imputer.fit_transform(df[['Adult Mortality']])
df['Alcohol'] = imputer.fit_transform(df[['Alcohol']])
df['Hepatitis B'] = imputer.fit_transform(df[['Hepatitis B']])
df[' BMI '] = imputer.fit_transform(df[[' BMI ']])
df['under-five deaths '] = imputer.fit_transform(df[['under-five deaths ']])
df['Total expenditure'] = imputer.fit_transform(df[['Total expenditure']])
df['Polio'] = imputer.fit_transform(df[['Polio']])
```

```
df['Diphtheria '] = imputer.fit_transform(df[['Diphtheria ']])
df['GDP'] = imputer.fit_transform(df[['GDP']])
df['Population'] = imputer.fit_transform(df[['Population']])
df[' thinness  1-19 years'] = imputer.fit_transform(df[[' thinness  1-19 years']])
df[' thinness 5-9 years'] = imputer.fit_transform(df[[' thinness 5-9 years']])
df['Income composition of resources'] = imputer.fit_transform(df[['Income compositi
df['Schooling'] = imputer.fit_transform(df[['Schooling']])
```

In [10]:
```
# verifying that there are no more null values in dataframe
df.isnull().sum()
```

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

Outliers

In this part, we will look for outliers (extreme values) in the dataset - this values can move results in one or the other direction.

In [11]:
```
# We find if our dataset contains outliers - not for categorical data
df.describe()[['Year', 'Life expectancy ', 'Adult Mortality','infant deaths','Alcoh
```

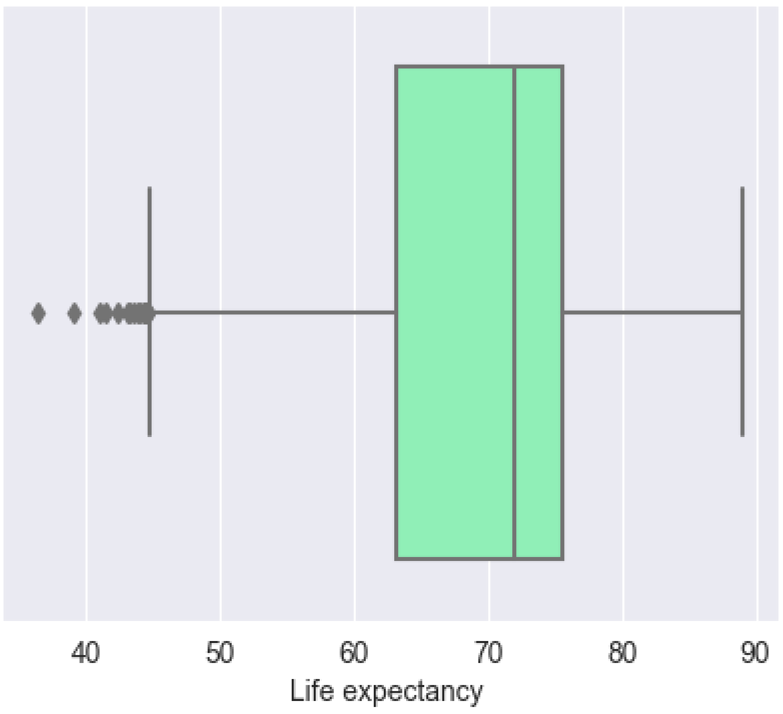
Out[11]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	He
count	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000	2938.000000	293
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	8
std	4.613841	9.507640	124.080302	117.926501	3.916288	1987.914858	2
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	
25%	2004.000000	63.200000	74.000000	0.000000	1.092500	4.685343	8
50%	2008.000000	72.000000	144.000000	3.000000	4.160000	64.912906	8
75%	2012.000000	75.600000	227.000000	22.000000	7.390000	441.534144	9
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	9

Visualization techniques for finding out outliers - Boxplot

Boxplot is great visualization technique for determining if our column has some outliers - in the main box, we can see values between 25th (Q1) and 75th (Q3) percentile, the line visualize median - this box is called Interquartile range (IQR). We determine which values are outliers by determining "minimum" (Q1 - 1.5 * IQR) and "maximum" (Q3 + 1.5 * IQR).

In [12]:
```
# demonstrating boxplot on column Life expectancy

plt.figure(figsize=(5,4))
sns.boxplot(x='Life expectancy ',data=df, palette='rainbow')
plt.show()
```



In [13]:
```
# The same can be done on our dataset for getting exact values of our outliers
# also demonstrated on our column Life expectancy
def find_outliers_IQR(df):
   q1=df.quantile(0.25)
   q3=df.quantile(0.75)
   IQR=q3-q1
   outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
   return outliers

find_outliers_IQR(df['Life expectancy '])
```

Out[13]:
```
1127    36.3
1484    44.5
1582    44.6
1583    44.0
1584    43.5
1585    43.1
2306    44.3
2307    43.3
2308    42.3
2309    41.5
2311    41.0
2312    39.0
2920    44.6
2921    43.8
2932    44.6
2933    44.3
2934    44.5
Name: Life expectancy , dtype: float64
```

Solving outliers

There are also multiple possibilities how to solve outliers, for example we can:

- remove them - drop the rows containing outliers
- cap them - for example, if we decide that max cap will be mean + 3*std, than everything above will be set to this mean
- replace them - as if they are missing values

For now, I won't be manipulating anyhow with this missing values.

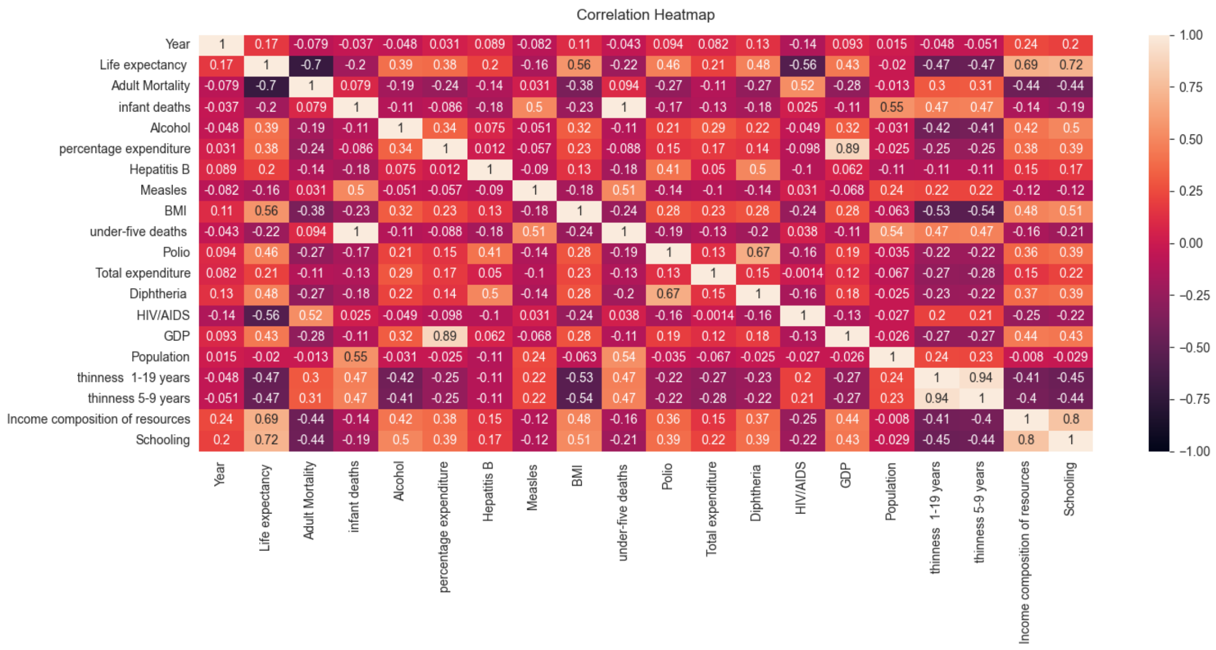
Solving categorical data

As I decided to use linear regression as our model, this algorithm can not work with not-numerical data. Because of that, we need to transform this columns (Year, Status) to something, with which our model could work.

In [14]:
```
# getting dummies
country_dummy = pd.get_dummies(df['Country'])
status_dummy = pd.get_dummies(df['Status'])
```

```
# dropping original columns
df.drop(['Country', 'Status'], inplace=True, axis=1)
```

In [15]:
```
# next we should concatenate  our dataset with dummies values, but first, we will v

# visualizing correlation
plt.figure(figsize=(16, 6))
# Store heatmap object in a variable to easily access it when you want to include m
# Set the range of values to be displayed on the colormap from -1 to 1, and set the
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True)
# Give a title to the heatmap. Pad defines the distance of the title from the top o
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



Based on this correlation heat-map, we can see strong (> 0.6) positive correlation between:

- under-five deaths and infant deaths (1)
- thinness 1-19 years and thinness 5-9 years (0.94)
- GDP and Percentage expenditure (0.89)
- Income composition of resources and Schooling (0.8)
- Schooling and Life expectancy (0.72)
- Income composition of resources and Life expectancy (0.69)
- Diptheria and Polio (0.67)

In [16]:
```
# concatenating dummy columns with our original dataset
df = pd.concat([df, country_dummy, status_dummy], axis=1)
df.head(10)
```

Out[16]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI
0	2015	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1
1	2014	59.9	271.0	64	0.01	73.523582	62.0	492	18.6
2	2013	59.9	268.0	66	0.01	73.219243	64.0	430	18.1
3	2012	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6
4	2011	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2
5	2010	58.8	279.0	74	0.01	79.679367	66.0	1989	16.7
6	2009	58.6	281.0	77	0.01	56.762217	63.0	2861	16.2
7	2008	58.1	287.0	80	0.03	25.873925	64.0	1599	15.7
8	2007	57.5	295.0	82	0.02	10.910156	63.0	1141	15.2
9	2006	57.3	295.0	84	0.03	17.171518	64.0	1990	14.7

10 rows × 215 columns

Before starting to create our models, we will split our dataset to train and test.

In [17]:
```
# our dependant variable (variable which we will be estimating with our models) wil
y = df['Life expectancy ']
x = df.drop(['Life expectancy '], axis=1)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_st
```

Linear Regression

Now that we know some things about our dataset and we prepared it, we move to more complicated part and learn machine learning models on this dataset.

We use Linear Regression if we assume linear relationship between dependent variable y and one (or more) independent variables x. In linear regression we try to find the best values for coeficients in our equation. For this we use method of Ordinary least squares.

Simple Linear Regression

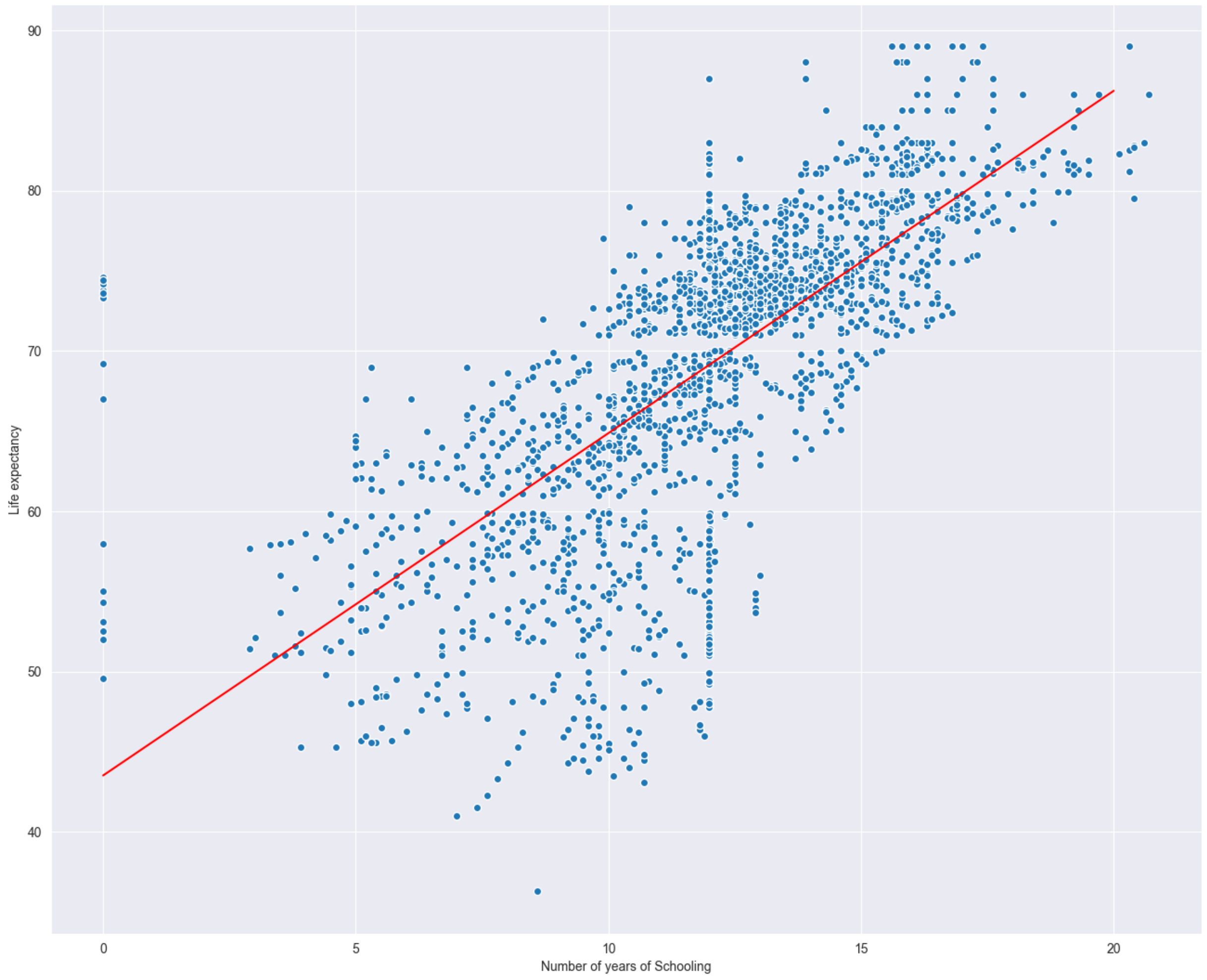
Simple linear regression means, that dependant variable y depends only on one independant variable x. Like in every type of linear regression, we are trying to find coeficients b0 and b1 for equation:

```
y = b0 + b1 * x
```

The value of b0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It’s the value of the estimated response y for x = 0. The value of b1 determines the slope of the estimated regression line.

In [18]:
```
# We will use two columns: Life expectancy as our dependant variable and Schooling
x = X_train['Schooling']

lr_model = LinearRegression()
lr_model.fit(x.values.reshape(-1,1), y_train)
print('Intercept: ' + str(lr_model.intercept_))
print('Beta coeficients (slope): ' + str(lr_model.coef_))

# Example of predicting values:
print("#" * 100)
print("Examples of predicting values using our trained model: ")
print('Number of years of Schooling = 7, Predicted life expectancy = ' + str(lr_mod
print('Number of years of Schooling = 5.5, Predicted life expectancy = ' + str(lr_m
print('Number of years of Schooling = 23, Predicted life expectancy = ' + str(lr_mo
print("#" * 100)

# visualizing the model and data
x_arr = np.arange(min(x), max(x)).reshape(-1,1)
plt.figure(figsize=(16, 13))

plt.scatter(x, y_train, edgecolors='white')

y_head = lr_model.predict(x_arr)
plt.plot(x_arr, y_head, color='red')
plt.xlabel('Number of years of Schooling')
plt.ylabel('Life expectancy')
plt.show()
```

```
Intercept: 43.52217241148958
Beta coeficients (slope): [2.13518872]
####################################################################################
################
Examples of predicting values using our trained model:
Number of years of Schooling = 7, Predicted life expectancy = 58.46849342502383
Number of years of Schooling = 5.5, Predicted life expectancy = 55.26571035069506
Number of years of Schooling = 23, Predicted life expectancy = 92.6315128845307
####################################################################################
################
```



Model evaluation

We can use 3 primary metrics for evaluating our linear model:

- Mean absolute error (MAE)
- Mean squared error (MSE)
- Root mean squared error (RMSE)

MAE: Average error **MSE**: similar to MAE, but noise is exaggerated. Results is not in base units. **RMSE**: similar to MSE, result is square rooted to make it more interpretable.

As each error, we want our model to give us lower values. We will use this error values in the end for comparing our model of simple linear regression and our model of multiple linear regression.

In [19]:
```
x_test = X_test['Schooling']
y_pred = lr_model.predict(x_test.values.reshape(-1,1))

print("MAE: " + str(metrics.mean_absolute_error(y_test.values, y_pred)))
print("MSE: "+ str(metrics.mean_squared_error(y_test.values, y_pred)))
print("RMSE: " + str(metrics.mean_squared_error(y_test.values, y_pred)**0.5))
```

```
MAE: 4.740327839464665
MSE: 41.635036035779216
RMSE: 6.452521680380409
```

Multiple Linear Regression (2 features)

It is a variant of linear regression, where we use two or more independent variables for calculating our dependent variable. For example, with two independent variables, our estimated regression function is:

```
y = b0 + b1*x1 + b2*x2
```

In [20]:
```
# We will use two columns: Life expectancy as our dependant variable and Schooling
x = X_train[['Schooling', 'Income composition of resources']]

lr_model = LinearRegression()
lr_model.fit(x.values, y_train)
print('Intercept: ' + str(lr_model.intercept_))
print('Beta coeficients (slope): ' + str(lr_model.coef_))
print("\n")

# Example of predicting values:
print("#" * 100)
print("Examples of predicting values using our trained model: ")
print('Number of years of Schooling = 7, Income composition of resources = 0.6, \n
print('Number of years of Schooling = 7, Income composition of resources = 0.3, \n
print('Number of years of Schooling = 5.5, Income composition of resources = 0.6, \
print('Number of years of Schooling = 23, Income composition of resources = 0.6, \n
print("#" * 100)

x_test = X_test[['Schooling', 'Income composition of resources']]
y_pred = lr_model.predict(x_test.values)

print("\n")
print("MAE: " + str(metrics.mean_absolute_error(y_test.values, y_pred)))
print("MSE: "+ str(metrics.mean_squared_error(y_test.values, y_pred)))
print("RMSE: " + str(metrics.mean_squared_error(y_test.values, y_pred)**0.5))
```

```
Intercept: 43.08110284987987
Beta coeficients (slope): [ 1.3661768  15.42593105]


####################################################################################
################
Examples of predicting values using our trained model:
Number of years of Schooling = 7, Income composition of resources = 0.6,
 Predicted life expectancy = 61.899899088804716
Number of years of Schooling = 7, Income composition of resources = 0.3,
 Predicted life expectancy = 57.27211977255137
Number of years of Schooling = 5.5, Income composition of resources = 0.6,
 Predicted life expectancy = 59.850633887429396
Number of years of Schooling = 23, Income composition of resources = 0.6,
 Predicted life expectancy = 83.75872790347479
####################################################################################
################


MAE: 4.3636232413543246
MSE: 37.6460394794977
RMSE: 6.135636843840882
```

Multiple linear regression (All features)

We can see that when we used 2 independent values (Schooling, Income composition of resources) to train our model, the value of error was decreasing - we trained better model. So, as the last model, we use for training our model of linear regression all features in our dataset.

In [23]:
```
# We will use two columns: Life expectancy as our dependant variable and Schooling
x = X_train

lr_model = LinearRegression()
lr_model.fit(x.values, y_train)
# print('Intercept: ' + str(lr_model.intercept_))
# print('Beta coeficients (slope): ' + str(lr_model.coef_))

x_test = X_test
y_pred = lr_model.predict(x_test.values)

print("\n")
print("MAE: " + str(metrics.mean_absolute_error(y_test.values, y_pred)))
print("MSE: "+ str(metrics.mean_squared_error(y_test.values, y_pred)))
print("RMSE: " + str(metrics.mean_squared_error(y_test.values, y_pred)**0.5))
```

```
MAE: 1.213982366962
MSE: 3.683311181186327
RMSE: 1.9191954515333574
```

Results

To summarize, in this notebook I prepared dataset and trained multiple Linear regression models. We can see, that the worst results gave the simple linear regression (doesn't mean it is bad, it would probably just need more data to give better results). When implementing multiple linear regression, we can see that the error was minimalizing - and, that means our model was getting better and better.

In the future, there could be implemented more machine learning models for predicting life expectancy based on this dataset, but this notebook is just an example how to implement simple data preparation and simple regression models.

By: Magdalena Ondruskova (maggie.ondruskova@gmail.com) Date: 03.05.2023