COSC2673 | Assignment 1

Introduction to Machine Learning Assignment

Table of Contents:

1. Goal Of the Project

The dataset used in this project is a modified real-world dataset, it entails records of devices and their life expectancy, collected during their follow-up period, where each device profile has its specific features. The goal of our project is to use the features in the dataset and predict the life expectancy using machine learning techniques and also find out which factors influence the life expectancy. Analyzing the output of the machine learning techniques applied and performing research on how to extend the modelling techniques is the end goal of the project. In our case target: life expectancy, train: rest of the features/columns. The machine learning algorithms we are using in this project to predict the life expectancy of the devices are polynomial regression: linear, ridge and lasso.

2. Introduction

As the goal of the project suggests, we aim to predict the life expectancy of the devices using the dataset by implementing machine learning algorithms such as polynomial regression: linear, ridge and lasso. To implement these algorithms on the dataset, we first need to perform data exploration.

3. Imports

Importing all the required and necessary libraries.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PowerTransformer


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
```

4. Dataset

Reading the dataset using the read_csv(), and then printing it to get an insight.

In [2]:
```
# reading and exploring the dataset
dataset = pd.read_csv('Data_Set.csv')
```

In [3]:
```
print(dataset)
```

```
        ID  TARGET_LifeExpectancy  Country  Year  Company_Status  \
0        1                   67.1      146  2017               0   
1        2                   59.8      146  2016               0   
2        3                   57.6      146  2015               0   
3        4                   55.5      146  2014               0   
4        5                   57.7      146  2013               0   
...    ...                    ...      ...   ...             ...   
2066  2067                   47.7       61  2006               0   
2067  2068                   46.0       61  2005               0   
2068  2069                   46.2       61  2004               0   
2069  2070                   47.1       61  2003               0   
2070  2071                   41.1       61  2002               0   

      Company_Confidence  Company_device_confidence  Device_confidence  \
0                    263                        262                264   
1                    271                        278                264   
2                    268                        246                290   
3                    272                        248                296   
4                    275                        278                272   
...                  ...                        ...                ...   
2066                 578                        596                560   
2067                  64                         66                 62   
2068                  69                         70                 68   
2069                 611                        574                648   
2070                 614                        574                654   

      Device_returen  Test_Fail  ...  ISO_23  TotalExpenditure  STRD_DTP  \
0                 62       0.01  ...       6              8.16      65.0   
1                 64       0.01  ...      58              8.18      62.0   
2                 66       0.01  ...      62              8.13      64.0   
3                 69       0.01  ...      67              8.52      67.0   
4                 71       0.01  ...      68              7.87      68.0   
...              ...        ...  ...     ...               ...       ...   
2066              36       2.46  ...      84              7.33      83.0   
2067              39       2.33  ...      85              8.18      83.0   
2068              41       2.44  ...      85              6.93      84.0   
2069              43       2.61  ...      86              6.56      85.0   
2070              44       2.62  ...      85              7.16      85.0   

      Engine_failure     GDP  Product_Quantity  Engine_failure_Prevalence  \
0                0.1  654.37          33736494                       17.2   
1                0.1  686.22            327582                       17.5   
2                0.1  707.55          31731688                       17.7   
3                0.1  750.35           3696958                       17.9   
4                0.1   71.16           2978599                       18.2   
...              ...     ...               ...                        ...   
2066            17.6   59.67          11731746                        7.2   
2067            18.2  480.66          11421984                        7.3   
2068            18.4  422.39            111249                        7.4   
2069            18.6  423.67           1824125                        7.4   
2070            18.7  382.99           1531221                        7.5   

      Leakage_Prevalence  IncomeCompositionOfResources        RD  
0                   17.3                         0.479  3.178050  
```

```
1                    17.5                        0.476  3.162278
2                    17.7                        0.470  3.146427
3                    18.0                        0.463  3.130495
4                    18.2                        0.454  3.082207
...                   ...                          ...       ...
2066                  7.1                        0.456  3.240370
2067                  7.2                        0.443  3.193744
2068                  7.3                        0.433  3.162278
2069                  7.4                        0.424  3.130495
2070                  7.5                        0.418  3.098387

[2071 rows x 24 columns]
```

5. Exploratory Data Analysis

5.1 Step 1 - Basic Insight

Using .shape on the dataset to view the total number of columns and rows the dataset entails.

Using info() function on the dataset to view basic details of each column in the Dataframe like Non-null, count, data type. Etc.

Using describe() function on the dataset to view basic statistical details of each column in the Dataframe like count, mean, std .etc.

As per our goal, we need to make predictions for the Life Expectancy, so we can explore the correlation of each column in the dataset with the Life Expectancy column.

In [4]:
```
dataset.shape
```

Out[4]:
```
(2071, 24)
```

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   ID                            2071 non-null   int64  
 1   TARGET_LifeExpectancy         2071 non-null   float64
 2   Country                       2071 non-null   int64  
 3   Year                          2071 non-null   int64  
 4   Company_Status                2071 non-null   int64  
 5   Company_Confidence            2071 non-null   int64  
 6   Company_device_confidence     2071 non-null   int64  
 7   Device_confidence             2071 non-null   int64  
 8   Device_returen                2071 non-null   int64  
 9   Test_Fail                     2071 non-null   float64
 10  PercentageExpenditure         2071 non-null   float64
 11  Engine_Cooling                2071 non-null   int64  
 12  Gas_Pressure                  2071 non-null   float64
 13  Obsolescence                  2071 non-null   int64  
 14  ISO_23                        2071 non-null   int64  
 15  TotalExpenditure              2071 non-null   float64
 16  STRD_DTP                      2071 non-null   float64
 17  Engine_failure                2071 non-null   float64
 18  GDP                           2071 non-null   float64
 19  Product_Quantity              2071 non-null   int64  
 20  Engine_failure_Prevalence     2071 non-null   float64
 21  Leakage_Prevalence            2071 non-null   float64
 22  IncomeCompositionOfResources  2071 non-null   float64
 23  RD                            2071 non-null   float64
```

```
dtypes: float64(12), int64(12)
memory usage: 388.4 KB
```

In [6]:
```python
# viewing the description of the dataset to get a deep insight
dataset.describe()
```

Out[6]:

	ID	TARGET_LifeExpectancy	Country	Year	Company_Status	Company_Co
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	207
mean	1036.000000	69.274505	95.360212	2009.518590	0.185418	16
std	597.990524	9.482281	54.861641	4.614147	0.388730	11
min	1.000000	37.300000	0.000000	2002.000000	0.000000	
25%	518.500000	63.000000	50.000000	2006.000000	0.000000	7
50%	1036.000000	71.200000	94.000000	2010.000000	0.000000	14
75%	1553.500000	76.000000	144.000000	2014.000000	0.000000	22
max	2071.000000	92.700000	192.000000	2017.000000	1.000000	69

8 rows × 24 columns

5.2 Step 2 - Correlation

Plotting a heatmap using seaborn to explore the correlation between each of the column's values.

1. If the correlation value is positive and closer to 1, it means that as one column increases so does the other.
2. If the correlation value is negative and closer to -1, it means that as one column increases, the other decreases.

- Columns positively correlated with Life Expectancy:

IncomeCompositionOfResources, RD, Gas_Presure, Company_Status, STRD_DTP, Test_Fail, GDP, ISO_23, PercentageExpenditure, TotalExpenditure, Year, Country

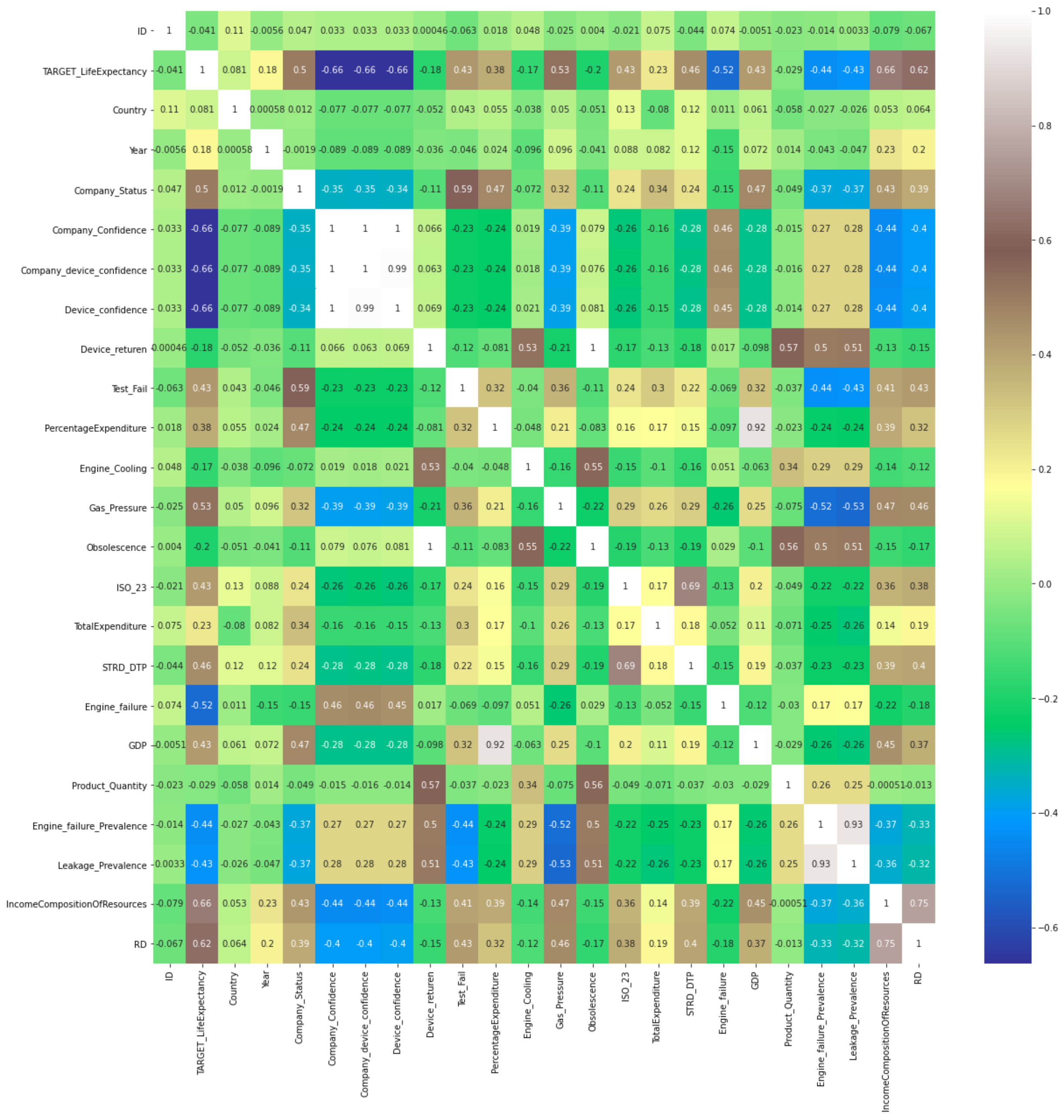
- Columns negatively correlated with Life Expectancy:

Product_Quantity, Engine_Cooling, Device_returen, Obsolescence, Leakage_prevalence, Engine_failure_Prevalence, Engine_failure, Device_confidence, Company_device_confidence, Company_Confidence

In [7]:
```python
# observing correlation relationship in the dataset
plt.figure(figsize=(20,20))
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(dataset.corr(), annot = True, cmap = 'terrain')
```

Out[7]:
```
<AxesSubplot:>
```

```
<Figure size 1440x1440 with 0 Axes>
```



```
In [8]:  cor = dataset.corr()
         cor['TARGET_LifeExpectancy'].sort_values(ascending=False)
```

```
Out[8]:  TARGET_LifeExpectancy           1.000000
         IncomeCompositionOfResources    0.659903
         RD                              0.624773
         Gas_Pressure                    0.531665
         Company_Status                  0.504971
         STRD_DTP                        0.457555
         Test_Fail                       0.433071
         GDP                             0.427650
         ISO_23                          0.426012
         PercentageExpenditure           0.378189
         TotalExpenditure                0.225359
         Year                            0.181780
         Country                         0.080722
         Product_Quantity               -0.028657
         ID                             -0.040947
         Engine_Cooling                 -0.168519
         Device_returen                 -0.179600
         Obsolescence                   -0.201468
         Leakage_Prevalence             -0.428562
         Engine_failure_Prevalence      -0.436399
         Engine_failure                 -0.522120
         Device_confidence              -0.661136
```

```
Company_device_confidence      -0.662934
Company_Confidence             -0.663425
Name: TARGET_LifeExpectancy, dtype: float64
```

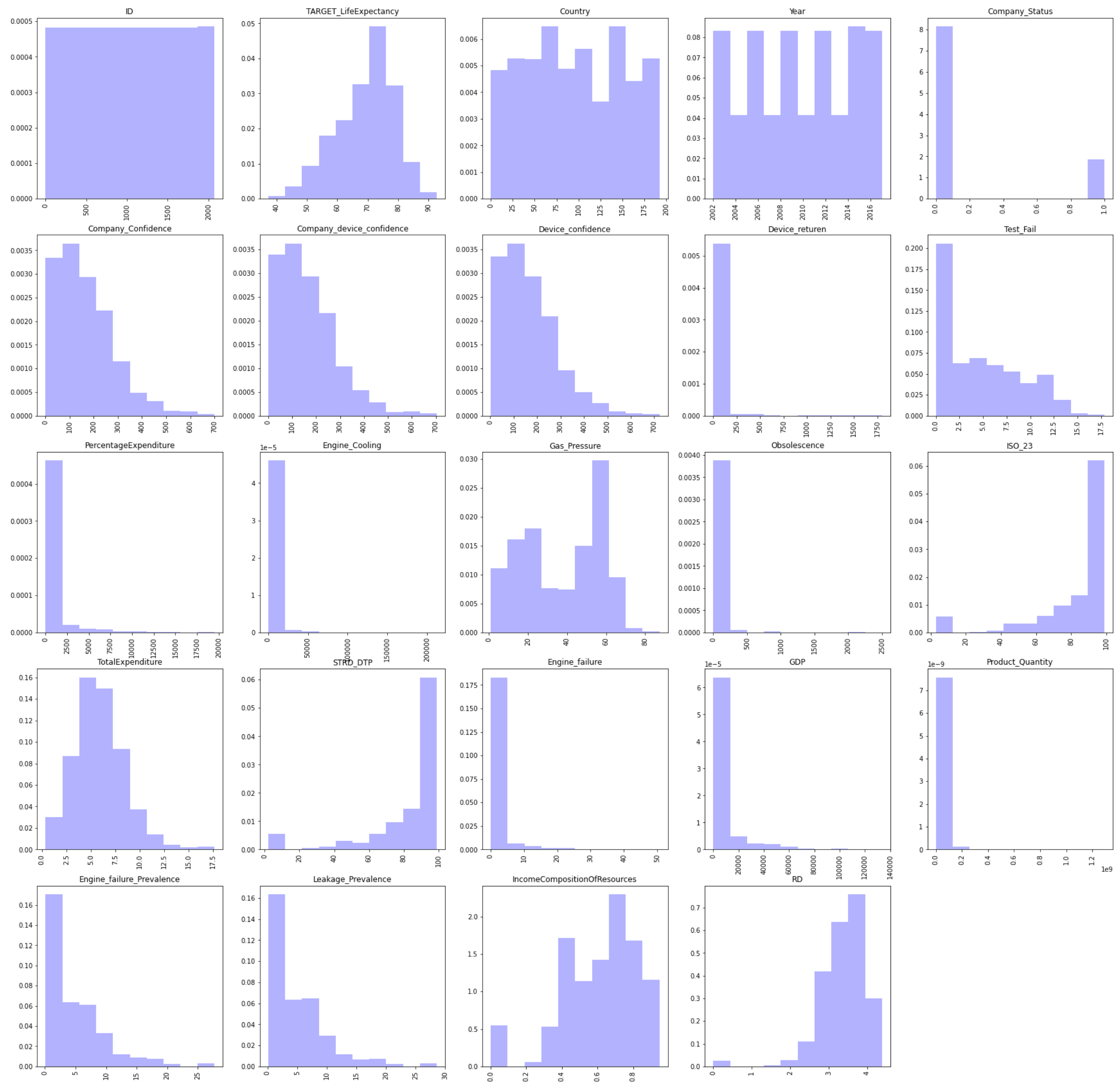
5.3 Estimating distribution of Variable using Histogram plot

- This is done to explore the distribution of each variable.
- In order to choose the most suitable statistical analysis to apply, we need to understand how the data are distributed. For example, Linear regression is a popular technique when the result variable is continuous (interval or ratio).

Key Observations:

- Company Status, Year and Country is a categorical variable.
- Rest of the variables are not categorical variables, and they are explored individually using box plot.
- Many attributes are heavily skewed, for example: Device return, percentage expenditure, engine cooling, Obsolescence, GDP, Product_Quantity, GDP, Engine failure etc.
- There are columns which have values far from the majority, like: ISO_23, STRD_DTP, RD. These columns need further investigation.
- Target variable - Life expectancy is distributed around 75, with minimum and maximum extremes of 40 and 90 respectively.

In [9]:
```python
# exploring distribution of variables using histogram
plt.figure(figsize=(30,30))
for i, col in enumerate(dataset.columns):
    plt.subplot(5,5,i+1, label=f"subplot{i}")
    plt.hist(dataset[col], alpha=0.3, color='b', density=True)
    plt.title(col)
    plt.xticks(rotation='vertical')
```



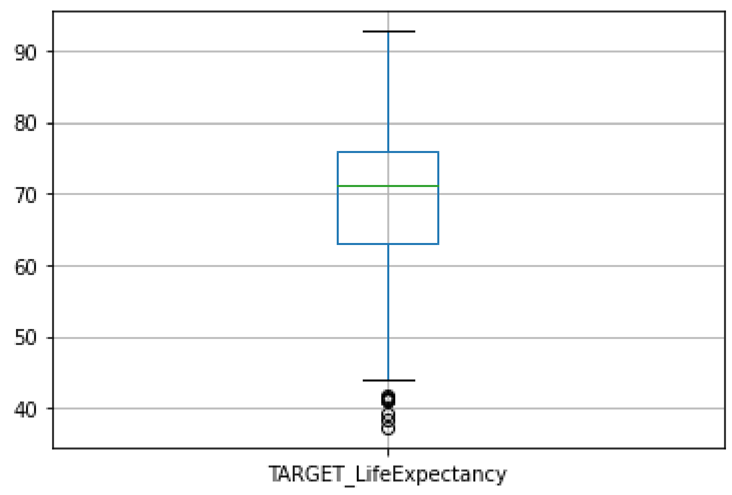
5.4 Exploring each column in the dataset

Target Life Expectancy Column

1. Exploring the target life expectancy column in the dataset using the Boxplot
2. Exploring the target life expectancy column will give us the Maximum, Minimum, Median, Quartiles of the life expectancy of devices in the dataset. By this we can get the statistical distribution of the column.

In [10]:
```
#exploring the target Life expectancy column will give us the Maximum, Minumum, Medi
dataset.boxplot(column=['TARGET_LifeExpectancy'], return_type='axes')
```

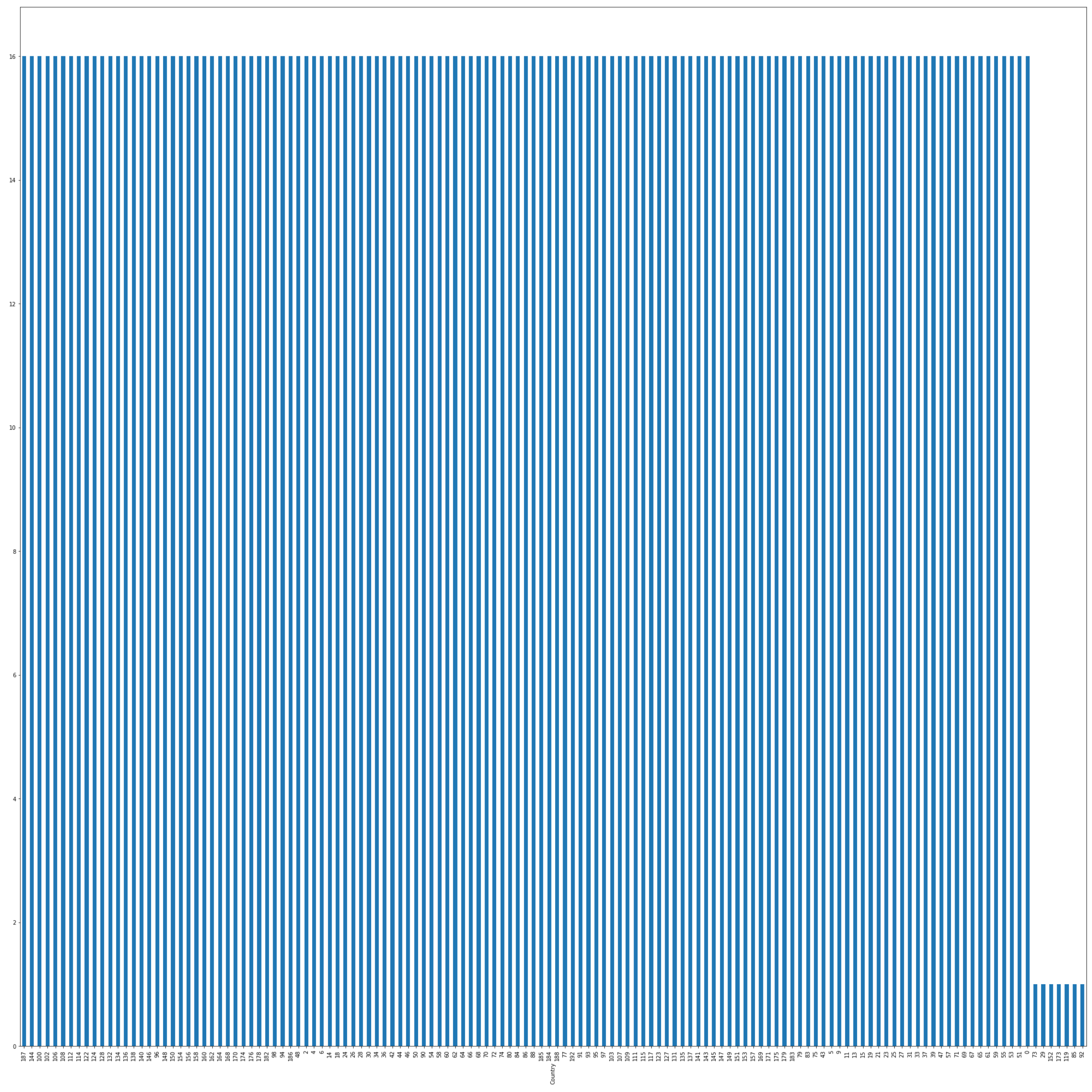
Out[10]:
```
<AxesSubplot:>
```



Company Column

1. Exploring the country column in the dataset using the Bar plot
2. Exploring the country column will give us the number of devices in each country, unique identifier of Country. By this we can get the count of the values.

In [11]:
```
# exploring the company/ country column will give us the number of devices under eac
plt.figure(figsize=(30,30))
ax = dataset['Country'].value_counts().plot(kind='bar')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Country', rotation=90)
plt.show()
```

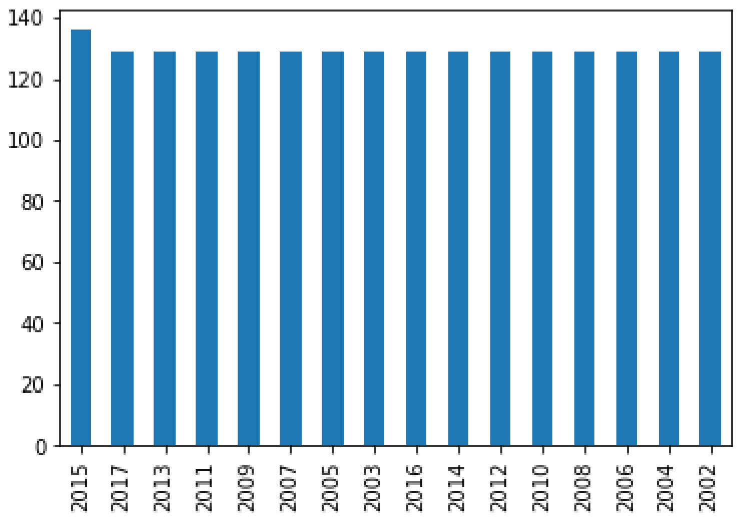


Year Column

1. Exploring the Year column in the dataset using the Bar plot.
2. Exploring the year column will give us the Year of devices in the dataset. By this we can get the count of the values.

In [12]:
```python
# exploring the year column will give us the number of devices under each year.
dataset['Year'].value_counts().plot(kind = 'bar')
```

Out[12]:
```
<AxesSubplot:>
```

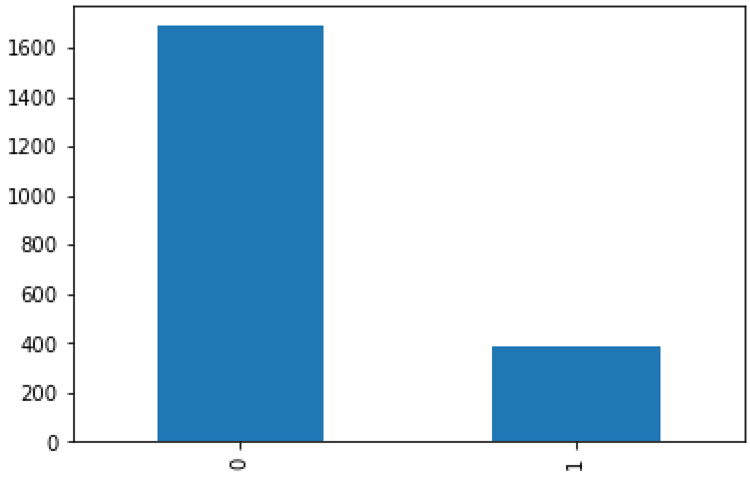


Company Status

1. Exploring the company status column in the dataset using the Bar plot
2. Exploring the company status column will give us the number of cases if the device is in developed or developing status (0, 1). By this we can get the count of the values.

In [13]:
```
# exploring the company status column will give us the data of devices under develop
dataset['Company_Status'].value_counts().plot(kind = 'bar')
```

Out[13]:
```
<AxesSubplot:>
```

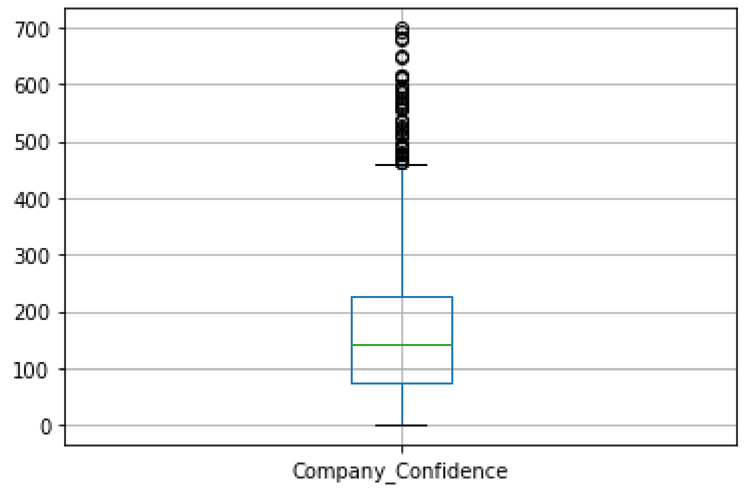


Company Confidence Column

1. Exploring the company confidence column in the dataset using the Boxplot
2. Exploring the company confidence column will give us the Maximum, Minimum, Median, Quartiles of the probability of any product from this company lasting between 15 and 60 Months per 1000 sample of devices in the dataset. By this we can get the statistical distribution of the column.

In [14]:
```
#exploring the company confidence column will give us the Maximum, Minumum, Median,
dataset.boxplot(column=['Company_Confidence'], return_type='axes')
```

Out[14]:
```
<AxesSubplot:>
```

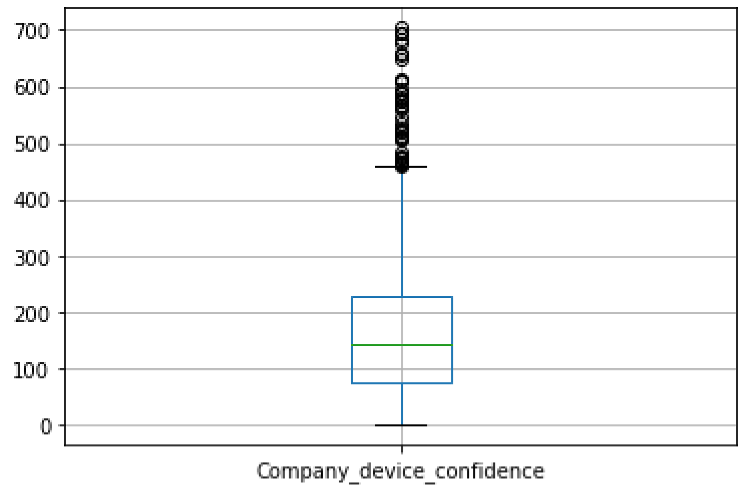


Company Device Confidence Column

1. Exploring the company device confidence column in the dataset using the Boxplot
2. Exploring the company device confidence will give us the Maximum, Minimum, Median, Quartiles of the probability of any product from this company lasting between 15 and 60 Months per 1000 sample of devices in the dataset. By this we can get the statistical distribution of the column.

In [15]:
```
#exploring the company device confidence column will give us the Maximum, Minumum, M
dataset.boxplot(column=['Company_device_confidence'], return_type='axes')
```

Out[15]:
```
<AxesSubplot:>
```

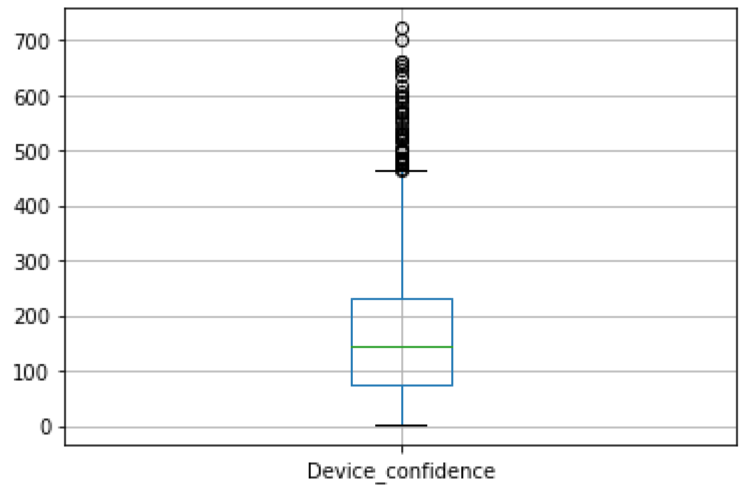


Device Confidence Column

1. Exploring the device confidence column in the dataset using the Boxplot
2. Exploring the device confidence column will give us the Maximum, Minimum, Median, Quartiles of the probability of any product from this company lasting between 15 and 60 Months per 1000 sample of devices in the dataset. By this we can get the statistical distribution of the column.

In [16]:
```
#exploring the device confidence column will give us the Maximum, Minumum, Median, Q
dataset.boxplot(column=['Device_confidence'], return_type='axes')
```

Out[16]:
```
<AxesSubplot:>
```

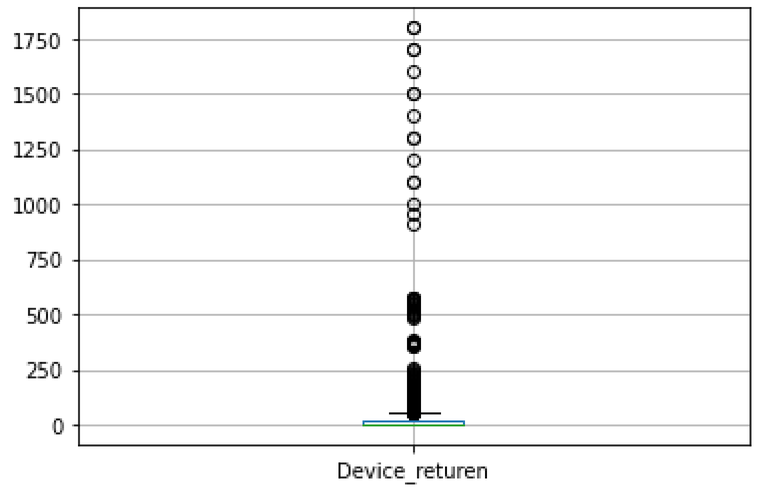


Device Return Column

1. Exploring the device return column in the dataset using the Boxplot.
2. Exploring the device return column will give us the Maximum, Minimum, Median, Quartiles of Number of returned devices based on faulty parts per 1000 samples of devices in the dataset. By this we can get the statistical distribution of the column.

In [17]:
```
#exploring the device return column will give us the Maximum, Minumum, Median, Quart
dataset.boxplot(column=['Device_returen'], return_type='axes')
```

Out[17]:
```
<AxesSubplot:>
```

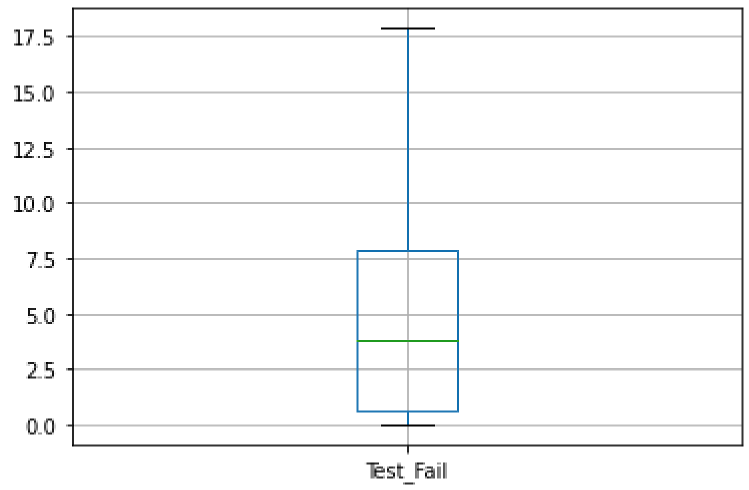


Test Fail Column

1. Exploring the test fail column in the dataset using Boxplot.
2. Exploring the test fail column will give us the Maximum, Minimum, Median, Quartiles of average of detected faulty products per week during sample inspections of devices in the dataset. By this we can get the statistical distribution of the column.

In [18]:
```
#exploring the test fail column will give us the Maximum, Minumum, Median, Quartiles
dataset.boxplot(column=['Test_Fail'], return_type='axes')
```

Out[18]:
```
<AxesSubplot:>
```

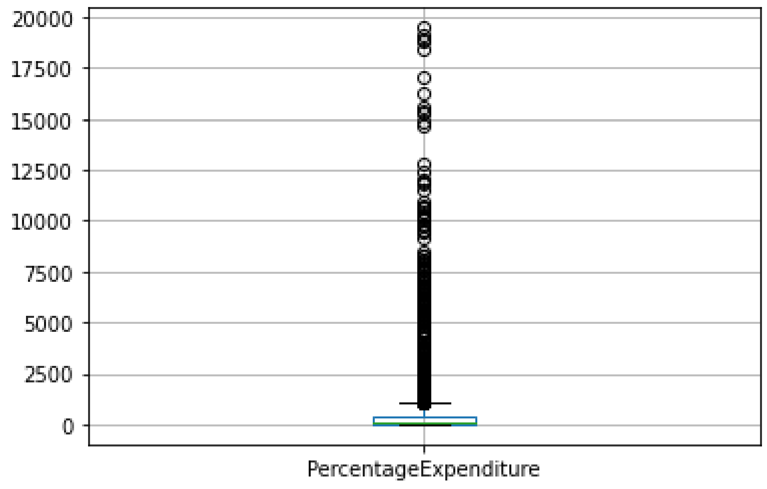


Percentage Expenditure Column

1. Exploring the percentage expenditure column in the dataset using Boxplot.
2. Exploring the percentage expenditure column will give us the Maximum, Minimum, Median, Quartiles of Expenditure on device's quality enhancement as a percentage of gross income from the product (%) in the dataset. By this we can get the statistical distribution of the column.

In [19]:
```
#exploring the percentage expenditure column will give us the Maximum, Minumum, Medi
dataset.boxplot(column=['PercentageExpenditure'], return_type='axes')
```

Out[19]:
```
<AxesSubplot:>
```

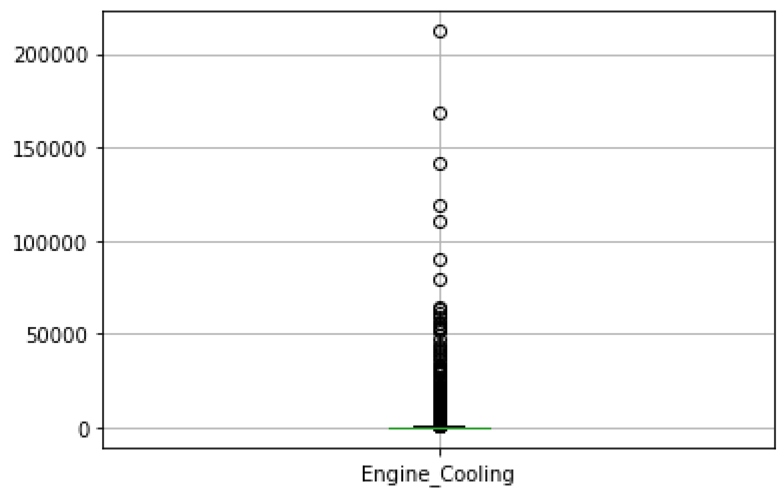


Engine Cooling Column

1. Exploring the Engine Cooling column in the dataset using Boxplot.
2. Exploring the Engine Cooling column will give us the Maximum, Minimum, Median, Quartiles of Cooling system failure- number of reported cases per 1000 sample of devices in the dataset. By this we can get the statistical distribution of the column.

In [20]:
```
#exploring the engine cooling column will give us the Maximum, Minumum, Median, Quar
dataset.boxplot(column=['Engine_Cooling'], return_type='axes')
```

Out[20]:
```
<AxesSubplot:>
```

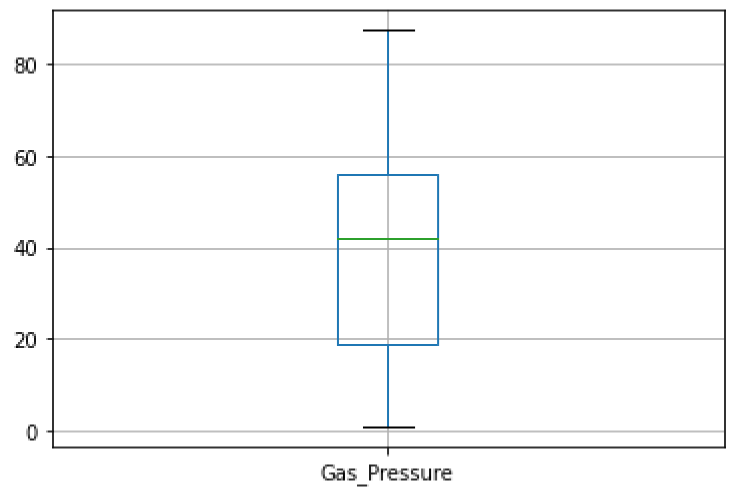


Gas Pressure

1. Exploring the Gas Pressure column in the dataset using Boxplot.
2. Exploring the Gas Pressure column will give us the Maximum, Minimum, Median, Quartiles of Average of gas pressure in section B of the device during rutine inspections on selected samples of devices in the dataset. By this we can get the statistical distribution of the column

```
In [21]: #exploring the gas pressure column will give us the Maximum, Minumum, Median, Quarti
         dataset.boxplot(column=['Gas_Pressure'], return_type='axes')
```

```
Out[21]: <AxesSubplot:>
```

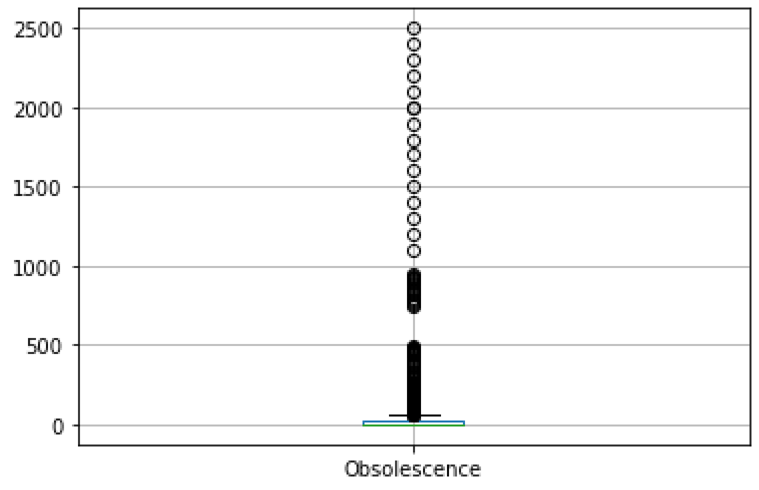


Obsolescence Column

1. Exploring the Obsolescence column in the dataset using Boxplot.
2. Exploring the Obsolescence column will give us the Maximum, Minimum, Median, Quartiles of under-five month obsolescence per 1000 sample of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [22]: #exploring the Obsolescence column will give us the Maximum, Minumum, Median, Quarti
         dataset.boxplot(column=['Obsolescence'], return_type='axes')
```

```
Out[22]: <AxesSubplot:>
```

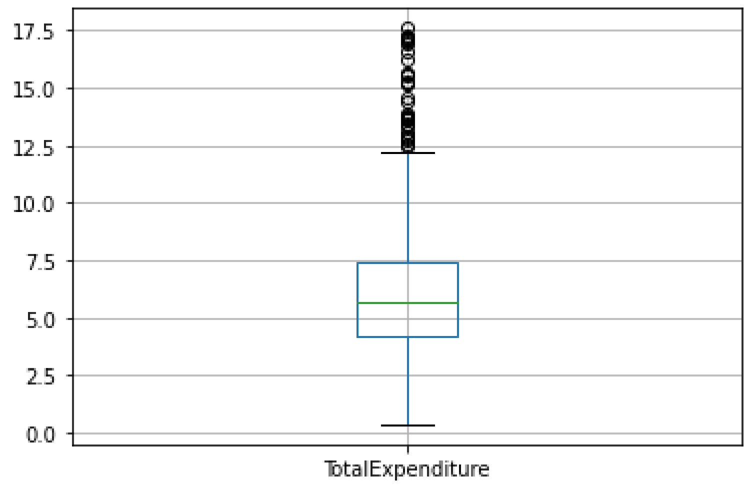


Total Expenditure Column

1. Exploring the Total Expenditure column in the dataset using Boxplot.
2. Exploring the Total Expenditure column will give us the Maximum, Minimum, Median, Quartiles of General government (of the company's location) expenditure on inspecting the standard of the products of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [23]: #exploring the total expenditure column will give us the Maximum, Minumum, Median, Q
         dataset.boxplot(column=['TotalExpenditure'], return_type='axes')
```

```
Out[23]: <AxesSubplot:>
```

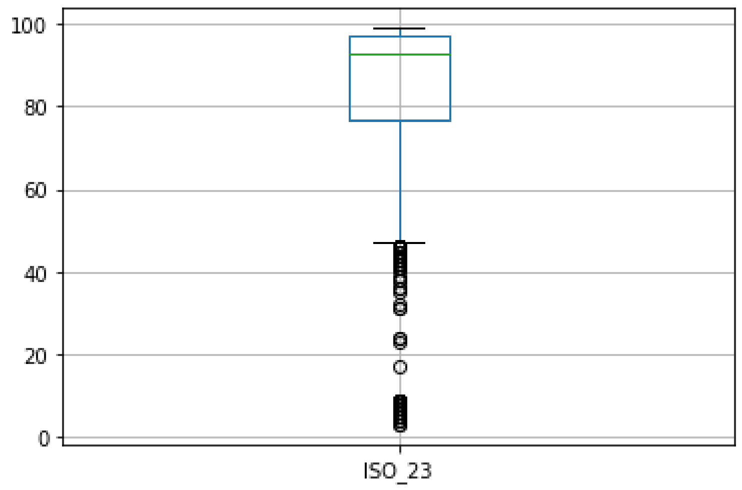


ISO 23 Column

1. Exploring the ISO 23 column in the dataset using Boxplot.
2. Exploring the ISO 23 column will give us the Maximum, Minimum, Median, Quartiles of Percentage of the product of the company meeting ISO_23 standard(%) of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [24]: #exploring the ISO 23 column will give us the Maximum, Minumum, Median, Quartiles of
         dataset.boxplot(column=['ISO_23'], return_type='axes')
```

```
Out[24]: <AxesSubplot:>
```

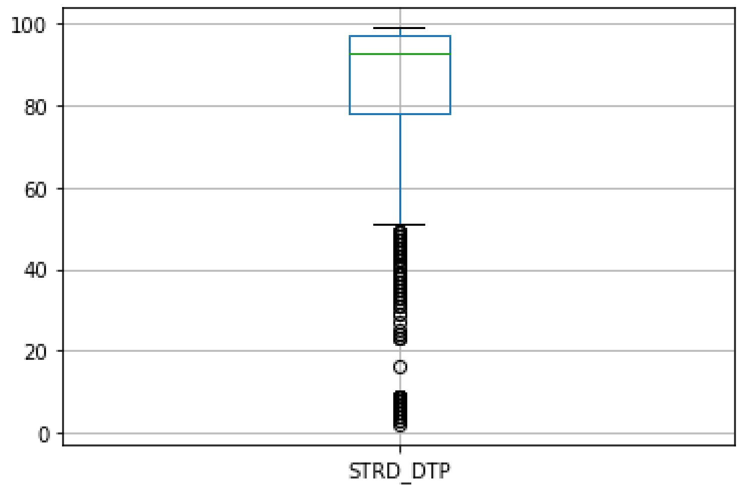


STRD_DTP Column

1. Exploring the STRD_DTP column in the dataset using Boxplot.
2. Exploring the STRD_DTP column will give us the Maximum, Minimum, Median, Quartiles of percentage of STRD_DTP covered warranty for 1-year of the device in the dataset. By this we can get the statistical distribution of the column.

In [25]:
```
#exploring the STRD_DTP column will give us the Maximum, Minumum, Median, Quartiles
dataset.boxplot(column=['STRD_DTP'], return_type='axes')
```

Out[25]:
```
<AxesSubplot:>
```

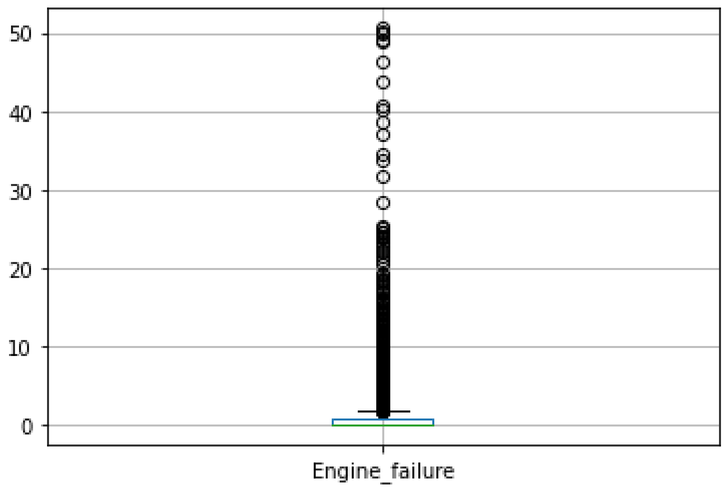


Engine Failure Column

1. Exploring the Engine Failure column in the dataset using Boxplot.
2. Exploring the Engine Failure column will give us the Maximum, Minimum, Median, Quartiles of Factor indicating the engine failure per 1 000 products (0-4 months) of the device in the dataset. By this we can get the statistical distribution of the column.

In [26]:
```
#exploring the engine failure column will give us the Maximum, Minumum, Median, Quar
dataset.boxplot(column=['Engine_failure'], return_type='axes')
```

Out[26]:
```
<AxesSubplot:>
```

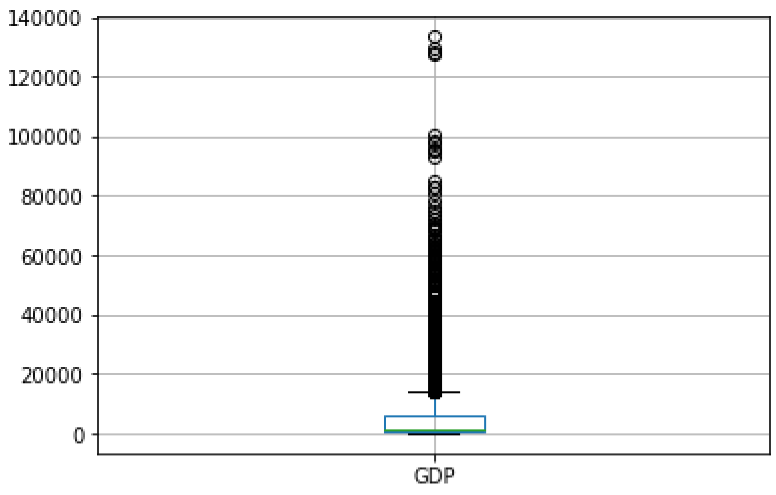


GDP Column

1. Exploring the GDP column in the dataset using Boxplot.
2. Exploring the GDP column will give us the Maximum, Minimum, Median, Quartiles of Gross Domestic Product per capita (in USD) where the company is located of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [27]: #exploring the GDP column will give us the Maximum, Minumum, Median, Quartiles of th
         dataset.boxplot(column=['GDP'], return_type='axes')
```

```
Out[27]: <AxesSubplot:>
```

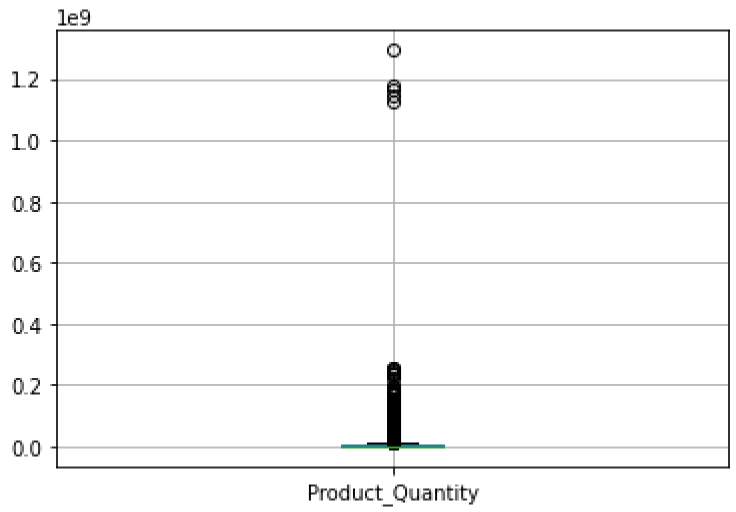


Product Quantity Column

1. Exploring the product quantity column in the dataset using Boxplot.
2. Exploring the product quantity column will give us the Maximum, Minimum, Median, Quartiles of Total product of the company per year of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [28]: #exploring the Product Quantity will give us the Maximum, Minumum, Median, Quartiles
         dataset.boxplot(column=['Product_Quantity'], return_type='axes')
```

```
Out[28]: <AxesSubplot:>
```

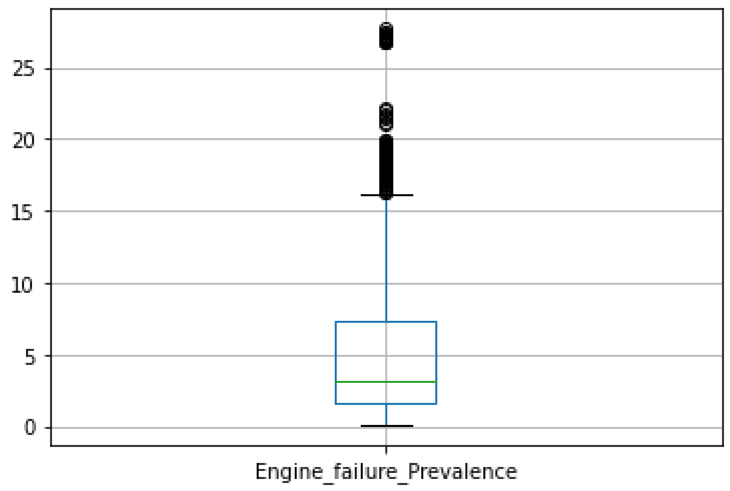


Engine Failure Prevalance Column

1. Exploring the Engine Failure Prevalence column in the dataset using Boxplot.
2. Exploring the Engine Failure Prevalence column will give us the Maximum, Minimum, Median, Quartiles of Prevalence of engine failure, in terms of the reduction of recorded failure compared to last year (%) of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [29]: #exploring the engine failure prevalance column will give us the Maximum, Minumum, M
         dataset.boxplot(column=['Engine_failure_Prevalence'], return_type='axes')
```

```
Out[29]: <AxesSubplot:>
```

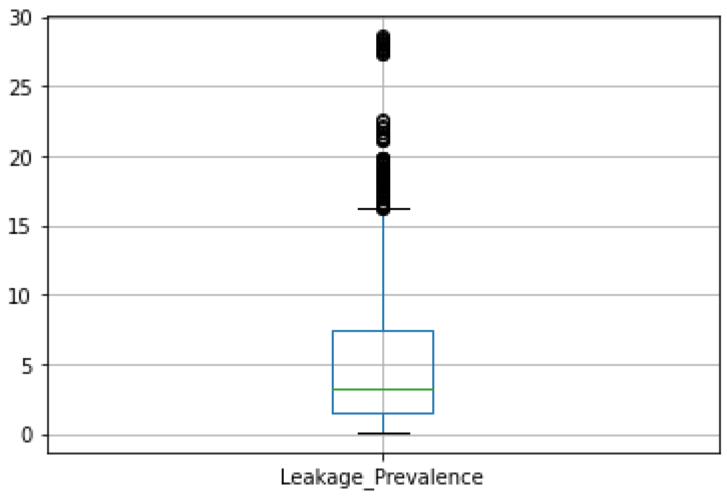


Leakage Prevalence Column

1. Exploring the Leakage Prevalence column in the dataset using Boxplot.
2. Exploring the Leakage Prevalence column will give us the Maximum, Minimum, Median, Quartiles of Prevalence of Leakage, , in terms of the reduction of recorded failure compared to last year (%) of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [30]: #exploring the leakage prevalence column will give us the Maximum, Minumum, Median,
         dataset.boxplot(column=['Leakage_Prevalence'], return_type='axes')
```

```
Out[30]: <AxesSubplot:>
```

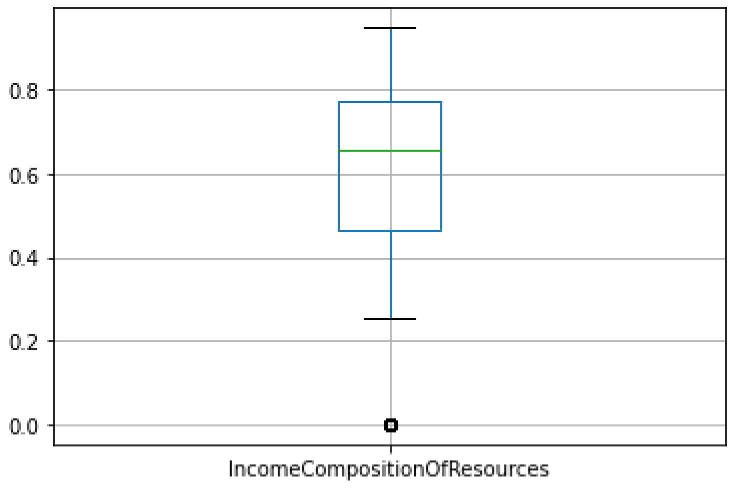


Income Composition Of Resources Column

1. Exploring the Income Composition of Resources column in the dataset using Boxplot.
2. Exploring the Income Composition of Resources column will give us the Maximum, Minimum, Median, Quartiles of Human Development Index in terms of income composition of resources of the device in the dataset. By this we can get the statistical distribution of the column.

```
In [31]: #exploring the income composition of resources column will give us the Maximum, Minu
         dataset.boxplot(column=['IncomeCompositionOfResources'], return_type='axes')
```

```
Out[31]: <AxesSubplot:>
```

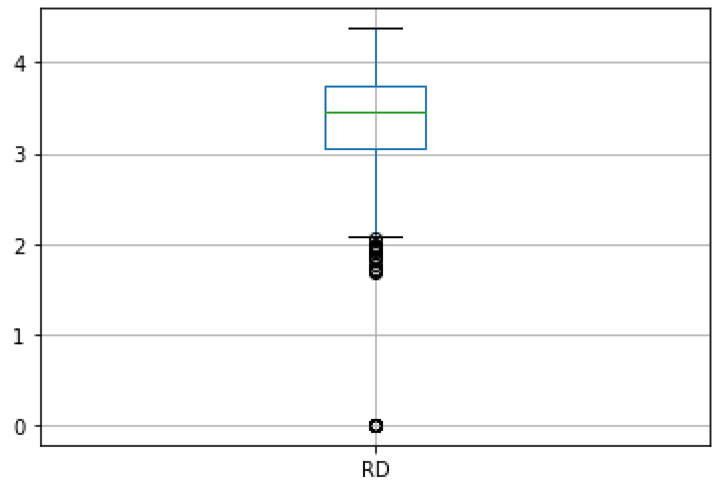


RD Column

1. Exploring the RD column in the dataset using Boxplot.
2. Exploring the RD column will give us the Maximum, Minimum, Median, Quartiles of Company's R&D sector of the device in the dataset. By this we can get the statistical distribution of the column

```
In [32]: #exploring the RD column will give us the Maximum, Minumum, Median, Quartiles of the
         dataset.boxplot(column=['RD'], return_type='axes')
```

```
Out[32]: <AxesSubplot:>
```



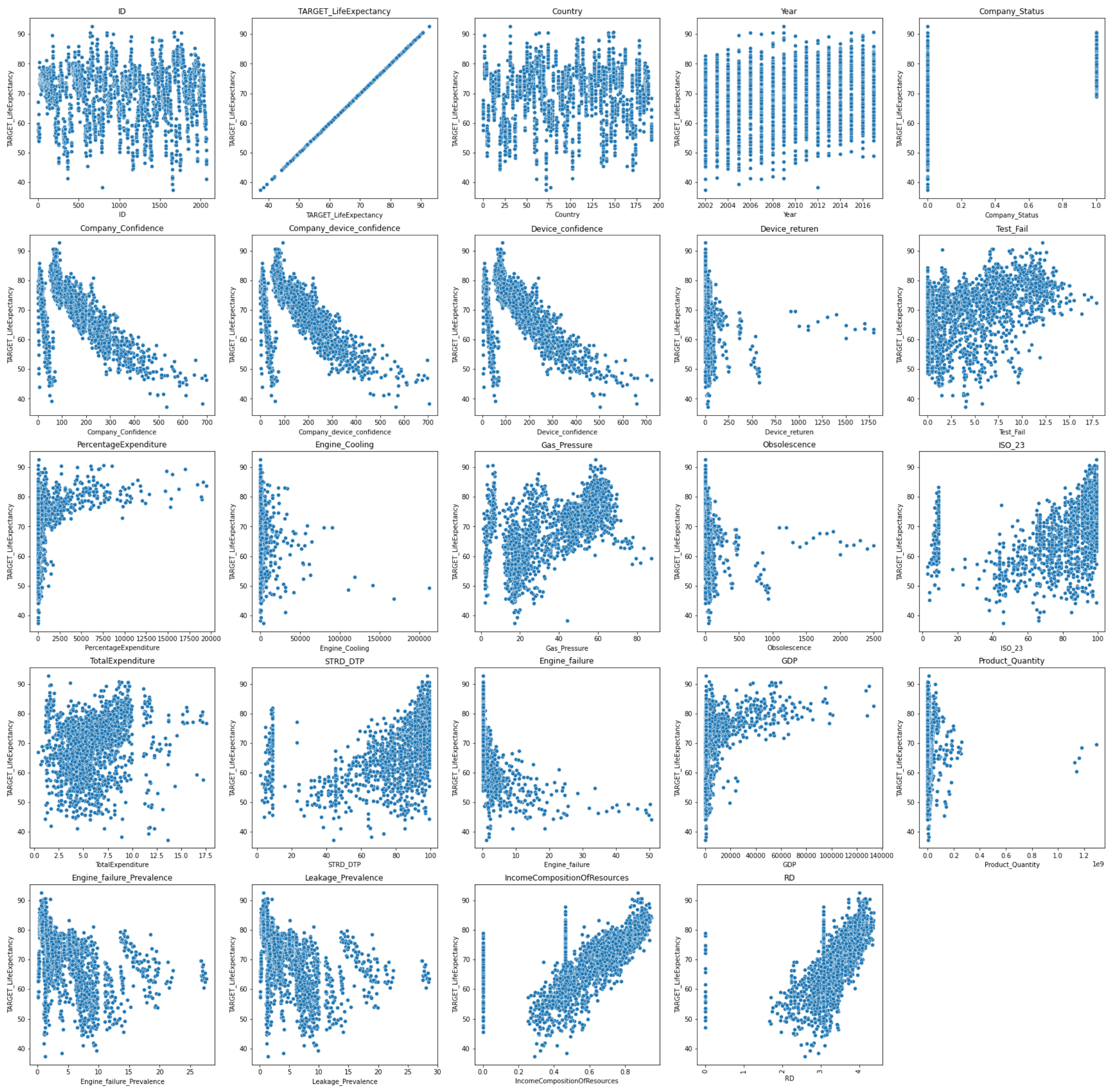
5.5 Relationship between Target Variable and other Variables

A linear or curvilinear additive relationship between a dependent variable and a group of independent variables can be used to explain the behaviour of a dependent variable. Therefore,

- a scatter plot is used to examine the relationship between the target variable - life expectancy and the rest of the other variables.
- To plot data and examine a fit for a linear regression model, we use the regplot() method. This is a very effecient and mutually exclusive method for estimating the regression model.

Scatter Plot

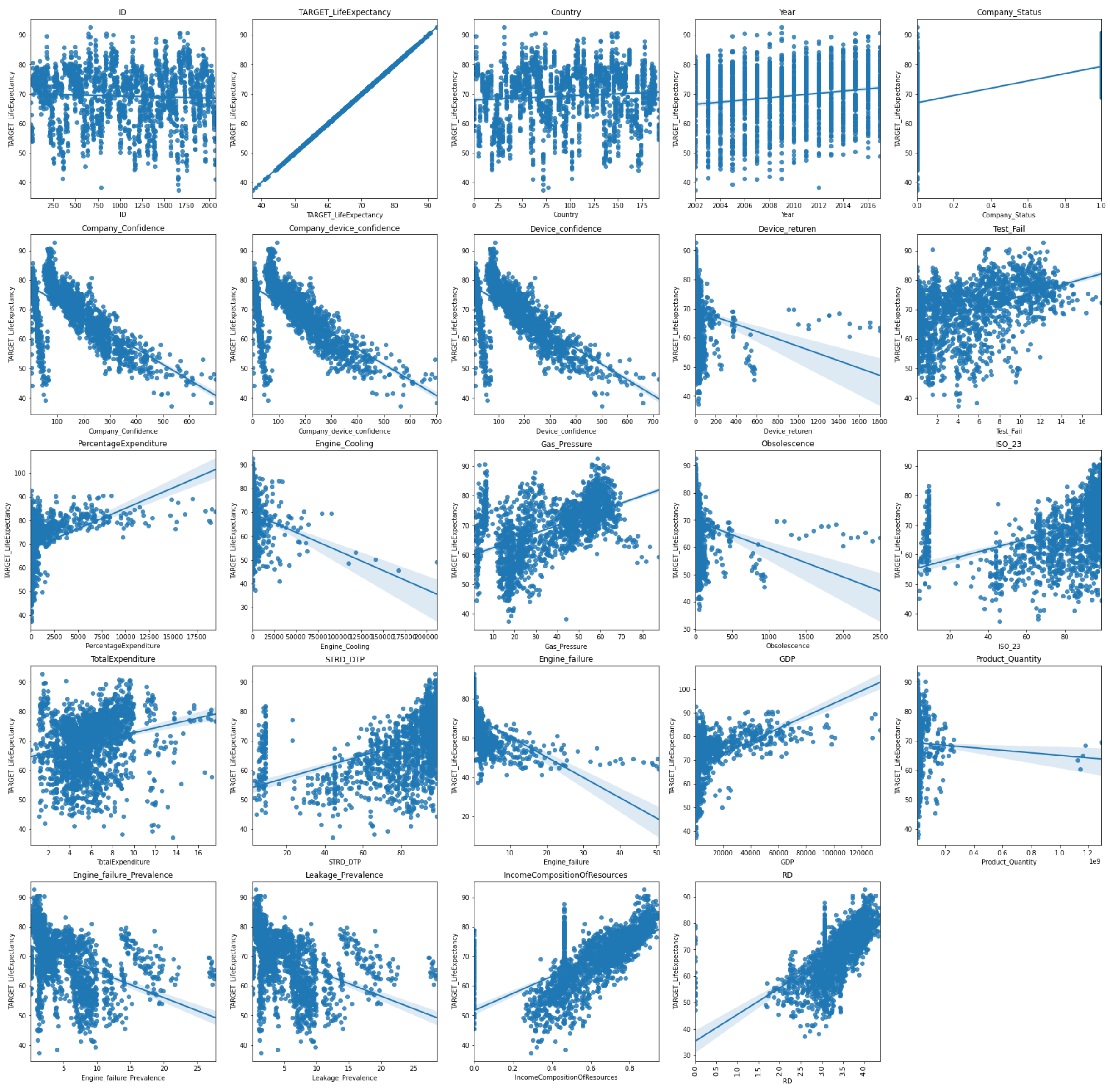
In [33]:
```
# Plotting a scatter plot to explore the relationship between target and other varia
import seaborn as sns
plt.figure(figsize=(30,30))
for i, col in enumerate(dataset.columns):
    plt.subplot(5,5,i+1)
    sns.scatterplot(data=dataset, x=col, y='TARGET_LifeExpectancy')
    plt.title(col)
plt.xticks(rotation='vertical')
plt.show()
```



- In the data set, the connection between the independent and dependent variables are not linear in multiple scenarios.
- Country, Year and Company Status are categorical variables and can be explore with box plot.
- The relationship between target variable and other variables are non-linear, such as - company confidence, company device confidence, device confidence. Etc. The relationship cannot be depicted using a straight line.
- Therefore Linear Regression, degree 1, cannot be used as a model for this specific dataset.
- From the EDA, we can conclude that a polynomial equation can be used to transform the relationship between the independent and target variable on a graph into a curvilinear relationship.
- The use of the line of greatest fit was the issue with linear regression. In other words, there must be a straight line where the scatterplots are located on a graph when we plot a dataset. The line of best fit is a curve in polynomial regression as opposed to a straight line in linear regression.

Regplot

In [34]:
```
# Plotting a scatter plot to determine the linear relationship with the help of a li
import seaborn as sns
plt.figure(figsize=(30,30))
for i, col in enumerate(dataset.columns):
    plt.subplot(5,5,i+1)
    sns.regplot(x=col,y='TARGET_LifeExpectancy', data=dataset)
    plt.title(col)
plt.xticks(rotation='vertical')
plt.show()
```



- In a line graph, however, non-linear data cannot be displayed. This is the reason why the graph is curved and there is no obvious relationship between the variables.
- The regplot provides us an insight and justifies that a line cannot be used to represent the relationship between the dependant and independant variable.
- There isn't a linear association between the variables, therefore, polynomial regression is performed.

6. Refining the Dataset for Modelling

- We remove the ID column from the dataset, as the ID column only decribes the unique identity of the device and this cannot be an attribute and is classified as row index. This will not create any impact on our machine learning model, therefore, we drop the column.

In [35]:
```
# Dropping ID column, as this is not a feature.
dataset = dataset.drop(columns='ID')
dataset.head()
```

Out[35]:

	TARGET_LifeExpectancy	Country	Year	Company_Status	Company_Confidence	Company_device_c
0	67.1	146	2017	0	263	
1	59.8	146	2016	0	271	
2	57.6	146	2015	0	268	
3	55.5	146	2014	0	272	
4	57.7	146	2013	0	275	

5 rows × 23 columns

- The Target LifeExpectancy column in our dataset serves as the target label, while the remaining columns are used to train the machine learning model. The Target LifeExpectancy column provides information about the device's life expectancy in months.
- Using the good train-test split, allocating 80% of the data for training and the remaining 20% for testing, and selecting random state = 0, which regulates the data shuffle applied prior to applying the split

In [36]:
```
dataset_X = dataset.drop(['TARGET_LifeExpectancy'], axis=1)
dataset_y = dataset['TARGET_LifeExpectancy']
```

In [37]:
```
# Splitting the dataset into train and test.
from sklearn.model_selection import train_test_split
dataset_X_train, dataset_X_test, dataset_y_train, dataset_y_test = train_test_split(
```

7. Feature Scaling

From the EDA we can observe that many of the columns are skewed and have outliers, while a few of them are categorical and a few of them do not have an even distribution. From the EDA, we have also come to a conclusion that we will be implementing polynomial regression in this model as per the relationship between the variables, and because to the sensitivity of Polynomial Regression to outliers, the existence of one or two outliers might potentially have a negative impact on results. Therefore we arrive at the decision of performing feature scaling on the columns.

Data must be scaled before being used in machine learning methods like linear regression, logistic regression, etc. that use gradient descent as an optimisation strategy.

We can observe that the following columns are categorical columns but are already in numerical format:-

- 'Country', 'Year', 'Company_Status'

The following features are already in numerical format, hence there is no need to encode the data. We however use MinMaxScaler, so the feature is returned with a default range of 0 to 1.

We can observe that the following columns have close to no outliers:-

- 'Test_Fail','IncomeCompositionOfResources','Gas_Pressure'

We thereby use MinMaxScaler on these features. It divides by the range after subtracting the feature's minimal value. The difference between the first maximum and initial minimum is called the range. MinMaxScaler keeps the original distribution's shape. The information present in the original data is not materially altered. The feature is returned with a default range of 0 to 1.

We can observe that the following columns are skewed and have outliers, which will affect the performance of the machine learning model:-

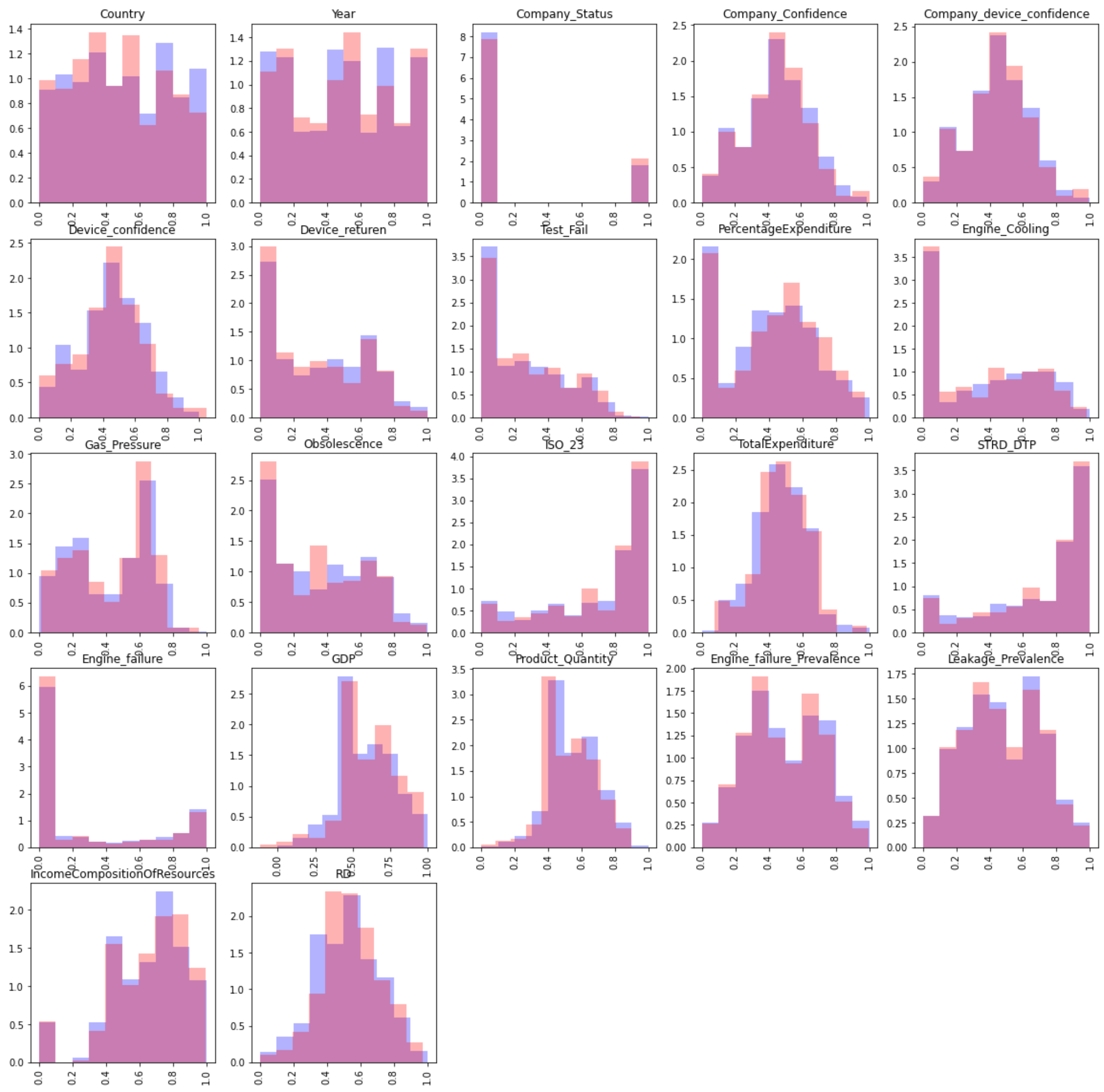
- 'Company_Confidence', 'Company_device_confidence', 'Device_confidence','Device_returen','PercentageExpenditure','Engine_Cooling','Obsolescence', 'ISO_23','TotalExpenditure', 'STRD_DTP', 'Engine_failure', 'GDP', 'Product_Quantity', 'Engine_failure_Prevalence', 'Leakage_Prevalence','RD'

Therefore, we use Power Transform on these variables as the power transform determines the ideal scaling factor to reduce skewness and stabilise volatility. PowerTransformer by default normalises the transformed output with a zero-mean, unit variance. In order to stabilise variance and reduce skewness, it offers non-linear transformations in which data is mapped to a normal distribution. We later use MinMaxScaler on the features after transformation, so the feature is returned with a default range of 0 to 1.

In [38]:
```python
# defining attributes for min max scaling
minmax_attributes = ['Test_Fail','IncomeCompositionOfResources','Gas_Pressure','Coun

# defining attributes for power transform
logNorm_attributes = ['Company_Confidence', 'Company_device_confidence', 'Device_con
       'Device_returen','PercentageExpenditure','Engine_Cooling','Obsolescence', 'IS
       'TotalExpenditure', 'STRD_DTP', 'Engine_failure', 'GDP',
       'Product_Quantity', 'Engine_failure_Prevalence', 'Leakage_Prevalence','RD']

dataset_X_train_scaled = dataset_X_train.copy()
dataset_X_test_scaled = dataset_X_test.copy()

# performing min max scaling on the min max attributes
minmaxscaler = MinMaxScaler().fit(dataset_X_train_scaled.loc[:, minmax_attributes])
dataset_X_train_scaled.loc[:, minmax_attributes] = minmaxscaler.transform(dataset_X_
dataset_X_test_scaled.loc[:, minmax_attributes] = minmaxscaler.transform(dataset_X_t

# performing power transform on the log norm attributes
powertransformer = PowerTransformer(method='yeo-johnson', standardize=False).fit(dat
dataset_X_train_scaled.loc[:, logNorm_attributes] = powertransformer.transform(datas
dataset_X_test_scaled.loc[:, logNorm_attributes] = powertransformer.transform(datase

# performing min max scaling on the log norm attributes
minmaxscaler_pt = MinMaxScaler().fit(dataset_X_train_scaled.loc[:, logNorm_attribute
dataset_X_train_scaled.loc[:, logNorm_attributes] = minmaxscaler_pt.transform(datase
dataset_X_test_scaled.loc[:, logNorm_attributes] = minmaxscaler_pt.transform(dataset
```

We can use plots to see if everything is in order and are identically distributed.

In [39]:
```python
# Exploring if everything is in order after splitting and scaling
plt.figure(figsize=(20,20))
for i, col in enumerate(dataset_X_train_scaled.columns):
    plt.subplot(5,5,i+1)
    plt.hist(dataset_X_train_scaled[col], alpha=0.3, color='b', density=True)
    plt.hist(dataset_X_test_scaled[col], alpha=0.3, color='r', density=True)
    plt.title(col)
    plt.xticks(rotation='vertical')
```



8. Model 1 - After Feature Scaling

- A straightforward linear regression approach can only be used when there is a linear relationship between the variables. Nevertheless, because our data are not linear, linear regression will be unable to provide a best-fit line.
- In order to solve this issue, we opt for polynomial regression, which reveals the curvilinear relationship between independent and dependent variables.
- A hyperparameter is the degree of order to be used. Using a high degree of polynomial tries to overfit the data, and using a low degree of polynomial tries to underfit the data. So we choose an ideal degree.

- To create a linear equation, we will first create the data and then apply simple linear regression on it. The performance of both algorithms in actual use may then be easily compared by applying polynomial regression on top of it.

8.1 Simple Linear Regression Model

In [40]:
```
# Simple linear regression model
model = LinearRegression()

# Fitting the training data to the model
model.fit(dataset_X_train_scaled, dataset_y_train)

# Using test set to make the predictions
dataset_y_pred = model.predict(dataset_X_test_scaled)

# Calculate R-squared and the MSE of the model
mse = mean_squared_error(dataset_y_test, dataset_y_pred)
r2 = r2_score(dataset_y_test, dataset_y_pred)

# Using Mean square error and r2 score to evaluate the performance of the model
print("MSE: {:.3f}".format(mse))
print("R-squared: {:.3f}".format(r2))
```

```
MSE: 20.112
R-squared: 0.763
```

- As a result of the equation we utilised, which has an intercept of 2, translating the input into polynomial terms by utilising the degree as 2. We select the degree by heat and try approach while addressing difficulties in the actual world.
- We fit the linear regression that is now operating as a polynomial regression after converting to polynomial terms. If you look at the Mean Square Error and R-squared value, the model is now functioning quite well. The model can be referred to as an effecient model comparitively, as the MSE has been reduced to 11.720 from 20.112 and the R-squared value has increased from 0.763 to 0.862.
- A type of linear regression known as polynomial regression estimates the relationship as an nth-degree polynomial and is a specific instance of multiple linear regression.
- An nth-degree polynomial function is used in polynomial regression to represent the connection between the dependent and independent variables. The term "quadratic model" refers to a polynomial of degree two; "cubic model" refers to a polynomial of degree three. As you can see in the below code, we have degree = 2. This value is based on trail and error, we arrive at a conclusion that this is a quadratic model.

8.2 Polynomial Regression Model

In [41]:
```
# Polynomial regression Model, setting degree to 2.
poly_model = PolynomialFeatures(degree=2)
dataset_X_poly_train = poly_model.fit_transform(dataset_X_train_scaled)
dataset_X_poly_test = poly_model.fit_transform(dataset_X_test_scaled)
model = LinearRegression()

# Fitting the training data to the model
model.fit(dataset_X_poly_train, dataset_y_train)

# Using test set to make the predictions
dataset_y_pred = model.predict(dataset_X_poly_test)
```

```
# Calculate R-squared and the MSE of the model
mse = mean_squared_error(dataset_y_test, dataset_y_pred)
r2 = r2_score(dataset_y_test, dataset_y_pred)

# Using Mean square error and r2 score to evaluate the performance of the model
print("MSE: {:.3f}".format(mse))
print("R-squared: {:.3f}".format(r2))
```

```
MSE: 11.720
R-squared: 0.862
```

Code Reference

1. https://data36.com/polynomial-regression-python-scikit-learn/ Date Accessed: 25-03-2023
2. https://www.geeksforgeeks.org/python-implementation-of-polynomial-regression/ Date Accessed: 25-03-2023
3. https://towardsdatascience.com/machine-learning-polynomial-regression-with-python-5328e4e8a386 Date Accessed 25-03-2023
4. https://www.kaggle.com/code/jnikhilsai/cross-validation-with-linear-regression Date Accessed: 28-03-2023

9. Regularisation, Hyperparameter tuning and K-Fold Cross validation

After We fit the linear regression that is now operating as a polynomial regression after converting to polynomial terms, and you look at the Mean Square Error and R-squared value, the model had an improvement.

The model can be referred to as an effecient model comparitively, as the MSE has been reduced to 11.720 from 20.112 and the R-squared value has increased from 0.763 to 0.862. **However, we can identify that there is still scope for improvement in the r2 score of the model and minimising the mean square error, due to their complexity and big datasets, models often overfit since the testing error may increase despite the training error being extremely modest. This can be accomplished successfully through the process of regularisation.**

What is Regularisation?

The term "regularisation" describes methods for calibrating machine learning models to reduce the adjusted loss function and avoid overfitting or underfitting. We can properly fit our machine learning model on a particular test set using regularisation, which lowers the mistakes in the test set. Regularization improves the model's ability to generalise and enhances its ability to function with untested data. In addition to adding uncertainty or unpredictability to the learning method, regularisation also makes the model and algorithm simpler. In regularisation, we typically maintain the same number of features while lowering the coefficients' magnitudes. This will directly create and impact on the r2 score and mse of the model.

Regularisation can be classified into two types:-

- Lasso Regularisation (L1)
- Ridge Regularisation (L2)

9.A Methodology

You can see below implementation of the lasso (L1) and ridge (L2) regression models. In both of the implementations, we first define the model with the help of a pipeline.

Make Pipeline

- Using the help of scikit-make pipeline learn's function, which produces a pipeline object that performs a number of data transformations before fitting a final estimator, this line of code creates a machine learning model.
- Polynomial Features:- This preprocessing step uses the incoming data to produce polynomial features.
- Regression model:- Lasso/ Ridge, is a linear regression model that does feature selection to allow the solver to converge and regularises the coefficients

Param Grid

The param grid entails:

- polynomialfeatures__degree:- This hyperparameter regulates how much of a polynomial regression model's polynomial features are used. In this case, it can take on values of 1, 2, or 3.
- __ alpha:- In a Lasso and Ridge regression model, this hyperparameter regulates the regularisation strength. It accepts numbers from a logarithmic scale with 25 evenly spaced points, ranging from 10^-5 to 10^1.

With this aid, GridSearchCV is able to test every conceivable combination of hyperparameters and identify the one that produces the greatest cross-validation score. Finally, the best parameters, best score is printed with the output. The best parameters entails, the optimal lambda value and the best polynomial degree.

K Fold Cross Validation

Throughout the entire procedure, training and testing would be executed precisely once for each set (fold). It aids in preventing overfitting. We can design a generalized model by avoiding this using k-fold cross-validation.

In our situation, K=5, which means that we are performing the Train and Test on a dataset that has been divided into 5 folds. One fold is used for testing throughout each run, with the remaining folds being used for training and advancing through iterations. Each data point is used twice, once in the hold-out set and once in the training set (K-1). Hence, one fold will be used for testing and the other folds for training at least once during the complete iteration.

- n_splits: how many folds should be utilised for cross-validation. As the value in this instance is 5, the data will be divided into 5 folds.
- shuffle: a boolean value that indicates whether or not the data should be shunted before being divided into folds. The data will be randomly shuffled before being divided into folds in this instance because the setting is True.

- random_state: a number that is an integer that is utilised to start the random number generator. This makes it possible to reproduce results by making sure the same set of random integers is generated each time the code is executed.

Grid Search - Hyperparameter tuning

GridSearchCV is a method for adjusting hyperparameters to find the best values for a particular model. This function aids in fitting the model to training set by looping through predefined hyperparameters. Hence, from the list of hyperparameters, it helps us choose the best parameters in the end.

In our situation, GridSearchCV tries every possible combination of the dictionary-passed data and assesses the model for each one using the Cross-Validation approach. As a result, after employing this function, it makes it easy to determine the accuracy and loss for each set of hyperparameters and select the combination that offers the best performance.

- model: the to-be-trained and-adjusted machine learning model. This could be a LinearRegression instance from the scikit-learn estimator class.
- Param_grids: a list of dictionaries or a dictionary specifying the hyperparameter space to search in. The values in the dictionary are lists of potential values for each key, which corresponds to a hyperparameter of the model. In order to identify the combination of hyperparameters that yields the highest cross-validated performance, GridSearchCV will thoroughly search through all conceivable combinations of hyperparameters.
- cv: The cross-validation object to use for calculating the model's performance. As the KFold object was started in the preceding code line, the data will be divided into 5 folds using K-fold cross-validation in this instance.
- scoring: The metric to utilise when assessing the model's performance. The generally employed indicator for regression issues in this situation is the negative mean squared error (MSE).
- n_jobs: how many CPU cores should be used to execute concurrent cross-validation and parameter tuning. A value of -1 indicates that all cores are being used.

9.1 Model 2 - Lasso Regression

A L1 regularisation method is **lasso regression**. For a more accurate forecast, it is prefered over regression techniques. Shrinkage is used in this model. When data values shrink towards the mean, this is referred to as shrinkage.

The penalty for L1 regularisation is equal to the amount of the coefficient in absolute terms. With this type of regularisation, sparse models with few coefficients may be produced. It's possible that some coefficients will go to zero and be dropped from the model. Coefficient values are closer to zero when the penalties are higher (ideal for producing simpler models).

In [42]:
```
# Pipeline defining, and increasing the iterations to 10000
model = make_pipeline(PolynomialFeatures(), Lasso(max_iter=10000))

# for search over, we define the hyperparameters
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'lasso__alpha': np.logspace(-5, 1, num=25)
              }
```

```
# cross validation scheme definition
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Grid search
grid_search = GridSearchCV(model, param_grid=param_grid, cv=cv, scoring='neg_mean_sq
grid_search.fit(dataset_X_train_scaled, dataset_y_train)

# using test set to make predictions
y_test_pred = grid_search.predict(dataset_X_test_scaled)

# calculate the mean squared error (MSE) on the test set
test_mse = mean_squared_error(dataset_y_test, y_test_pred)

# calculate the R2 score on the test set
r2 = r2_score(dataset_y_test, y_test_pred)

# Best hyperparameters to be printed out
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", -grid_search.best_score_)

# Using Mean square error and r2 score to evaluate the performance of the model
print('MSE:', test_mse)
print('R-squared:', r2)
```

```
Best hyperparameters:  {'lasso__alpha': 0.0031622776601683794, 'polynomialfeatures__
degree': 3}
Best score:  11.292226875177231
MSE: 9.768856519457893
R-squared: 0.8850910210258136
```

Code Reference

1. https://www.kaggle.com/code/deepakdodi/lasso-and-ridge-hypertuning-over-gapminder-dataset/notebook Date Accessed: 28-03-2023
2. https://medium.com/mlearning-ai/lasso-regression-and-hyperparameter-tuning-using-sklearn-885c78a37a70 Date Accessed: 28-03-2023
3. https://machinelearningmastery.com/lasso-regression-with-python/ Date Accessed: 28-03-2023
4. https://www.kaggle.com/code/jnikhilsai/cross-validation-with-linear-regression Date Accessed: 28-03-2023

9.2. Model 3 - Ridge Regression

A L2 regularisation method is ridge regression. It imposes a penalty equal to the square of the coefficients' magnitude. Predicted values differ much from real values when the problem of multicollinearity arises, least-squares are unbiased, and variances are significant.

The "squared magnitude" of the coefficient is added to the loss function as a penalty term in ridge regression. The significance of predictors can be estimated using L2 regression, and based on that, the insignificant predictors can be penalised.

In [49]:
```
# Define the pipeline with Ridge regression
model = make_pipeline(PolynomialFeatures(), Ridge())

# for search over, we define the hyperparameters
param_grid = {'ridge__alpha': np.logspace(-5, 1, num=25),
              'polynomialfeatures__degree': [1, 2, 3]}

# cross validation scheme definition
```

```
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Grid search
grid_search = GridSearchCV(model, param_grid=param_grid, cv=cv, scoring='neg_mean_sq
grid_search.fit(dataset_X_train_scaled, dataset_y_train)

# using test set to make predictions
y_pred = grid_search.predict(dataset_X_test_scaled)

# Calculate the MSE and R-squared of the model on the test set
mse = mean_squared_error(dataset_y_test, y_pred)
r2 = r2_score(dataset_y_test, y_pred)

# Best hyperparameters to be printed out
print('Best hyperparameters:', grid_search.best_params_)
print("Best score: ", -grid_search.best_score_)

# Using Mean square error and r2 score to evaluate the performance of the model
print('MSE:', mse)
print('R-squared:', r2)
```

```
Best hyperparameters: {'polynomialfeatures__degree': 3, 'ridge__alpha': 1.7782794100
389228}
Best score:  11.37756398814004
MSE: 9.095857659843887
R-squared: 0.8930073632972967
```

Code Reference

1. https://stackoverflow.com/questions/57376860/how-to-run-gridsearchcv-with-ridge-regression-in-sklearn Date Accessed: 31-03-2023
2. https://machinelearningmastery.com/ridge-regression-with-python/ Date Accessed: 31-03-2023
3. https://datascience.stackexchange.com/questions/66389/ridge-regression-model-creation-using-grid-search-and-cross-validation Date Accessed: 31-03-2023
4. https://www.kaggle.com/code/deepakdodi/lasso-and-ridge-hypertuning-over-gapminder-dataset/notebook Date Accessed: 31-03-2023
5. https://machinelearninghd.com/ridgecv-regression-python/ Date Accessed: 31-03-2023
6. https://www.kaggle.com/code/jnikhilsai/cross-validation-with-linear-regression Date Accessed: 28-03-2023

10. Ultimate Judgement

10.1 Method:

- **Mean Square Error** : MSE is used to determine how closely predictions or estimates match actual values. The lower the MSE, the more accurate the forecast is. Regression models utilise this to evaluate their models, and a lower number denotes a better match. In our case, we use MSE to calculate the error in each model right from simple linear regression to the ridge model. We can see that the value of MSE has been reduced significantly from the first model to the last one.

- **R2 Score**: The data points' scatter around the fitted regression line is measured using R-squared. For multiple regression, it is also known as the coefficient of determination or the coefficient of multiple determination. Higher R-squared values for the same data set indicate less discrepancy between the fitted values and the observed data. In our case, we use r2 score to calculate the evaluation score in each model right from simple linear regression to the ridge model. We can see that the value of r2 score has been increased significantly from the first model to the last one.

- **Best Score**: The mean cross-validated score of the best estimator represents the best score_ from GridSearchCV. For instance, in our case, GridSearchCV divides the data into 5 folds and trains the model 5 times when using 5-fold cross-validation. Every time, it sets aside one fold and trains the model using the other four folds. The model's performance is then evaluated using the fold that was left out. The final score is the mean performance of the five models returned. We calculate the same for our Lasso and Ridge regression model and you can see that there is a slight imporvement, which is a positive sign.

10.2 Evaluation:

- From Mean Square error we can infer that lower the score, better the model effeciency.

1. Simple Linear Regression: 20.112
2. Polynomial Regression Model: 11.720
3. Lasso Regression Model: 9.768856519457893
4. Ridge Regression Model: 9.095857659843887

It is oberserved that the, MSE has been reduced significantly which is a good sign and so we can consider Ridge Regression Model from this scenario.

- From R2 square we can infer that higher the score, better the model effeciency.

1. Simple Linear Regression: 0.763
2. Polynomial Regression Model: 0.862
3. Lasso Regression Model: 0.8850910210258136
4. Ridge Regression Model: 0.8930073632972967

It is oberserved that the, R2 score has been increased significantly which is a good sign and so we can consider Ridge Regression Model from this scenario. It is also observed that, the r2 square score of Lasso and Ridge regression model is not very apart.

- From Best Score - grid search CV, we can infer that, higher the score, better the model effeciency.

1. Lasso Regression Model: 11.292226875177231
2. Ridge Regression Model: 11.37756398814004

A slight improvement is best score is considered as a favourable sign, therefore Ridge Regression takes a win over Lasso Regression.

Ridge Regression Model can be concluded as the best model for predicting the Life Expectancy of the device. With Best hyperparameters: {'polynomialfeaturesdegree': 3, 'ridgealpha': 1.7782794100389228}

11. Saving Predictions File

```
In [47]: # Reading and writing the predictions of Life expectancy into a csv file.
dummy_x = dataset_y_test.reset_index()
predictions = pd.DataFrame()
predictions['ID'] = dummy_x['index']
predictions['Target'] = y_test_pred
predictions.to_csv('s3828461.csv')
```

Report - References

1. https://m.tutorialspoint.com/seaborn/seaborn_regplot_method.htm#:~:text=regplot()%20metho Date Accessed: 18-03-2023

2. https://www.atoti.io/articles/when-to-perform-a-feature-scaling/ Date Accessed: 19-03-2023

3. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ Date Accessed: 19-03-2023
4. https://medium.com/analytics-vidhya/why-scaling-is-important-in-machine-learning-aee5781d161a Date Accessed: 19-03-2023
5. https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html Date Accessed: 22-03-2023

6. https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb Date Accessed: 22-03-2023

7. https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18 Date Accessed: 22-03-2023
8. https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/#:~:text=Polynomial%20regression%20is%20a%20form,convert%20it%20into%20Po Date Accessed: 22-03-2023

9. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a Date Accessed: 24-03-2023

10. https://www.analyticsvidhya.com/blog/2022/08/regularization-in-machine-learning/ Date Accessed: 24-03-2023

11. https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/ Date Accessed: 26-03-2023

12. https://www.mygreatlearning.com/blog/gridsearchcv/ Date Accessed: 26-03-2023
13. https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/#:~:text=Lasso%20regression%20is%20a%20regularization,i.e.%20models%20with%2 Date Accessed: 26-03-2023
14. https://www.mygreatlearning.com/blog/what-is-ridge-regression/#:~:text=Ridge%20regression%20is%20a%20model,away%20from%20the%20actual Date Accessed: 28-03-2023

15. https://machinelearningmastery.com/k-fold-cross-validation/ Date Accessed: 28-03-2023

16. https://www.mygreatlearning.com/blog/gridsearchcv/ Date Accessed: 28-03-2023

17. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html Date Accessed: 2-04-2023

18. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html Date Accessed: 2-04-2023

19. https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html Date Accessed: 2-04-2023