REGRESSION PROJECT ON PREDICTING LIFE EXPECTANCY USING WHO DATASET

Crash_Course_in_Statistical_Learning_Written_Section

Shreya Jaiswal

ABSTRACT

This study aimed to predict life expectancy using a dataset compiled by the World Health Organization (WHO). The dataset contained information on over 200 countries and included features such as healthcare spending, GDP, education, and other social and economic indicators.

The data was preprocessed by removing missing values, encoding categorical variables, and scaling numerical features. A linear regression model was then trained to predict life expectancy based on the selected features. The model was evaluated using metrics such as mean squared error, R-squared, and p-values to determine its accuracy and statistical significance.

Overall, the findings suggest that linear regression can be an effective method for predicting life expectancy based on social and economic indicators. The study has implications for healthcare policy and international development, as well as for individuals seeking to understand and improve their health outcomes.

ABOUT THE DATASET

The Global Health Observatory (GHO) data repository under World Health Organization (WHO) keeps track of the health status and other factors related to health for all countries. The dataset is made avalailable to the public for the purpose of performing health data analaysis. The data-set related to life expectancy, health factors for 193 countries has been collected from the WHO data repository website and its corresponding economic data was collected from United Nation website.

> dataset origin : https://www.kaggle.com/kumarajarshi/life-expectancy-who

---

> **PROJECT OVERVIEW**

> The project aims to perform Exploratory Data Analysis (EDA) on the WHO dataset and implement linear regression model to predict the Life Expectancy.

LIFE EXPECTANCY: INTRODUCTION

> **Life expectancy** is the statistical measure of the average number of years a person is expected to live based on various factors, such as their demographic, health, and social status. It is often used as a summary measure of the overall health and well-being of a population.

> Life expectancy can be calculated at birth, at specific ages, or for a given period of time. It is influenced by a range of factors, including genetics, lifestyle, and access to healthcare.

> Life expectancy varies widely across different countries and regions, with some countries having much higher life expectancies than others. Factors such as income, education, and healthcare access play a significant role in determining life expectancy.

> In general, life expectancy has been increasing globally over the past century, due in part to improvements in healthcare, sanitation, and nutrition. However, some countries and regions still face significant health challenges that can impact life expectancy, such as high rates of infectious diseases or inadequate healthcare systems.

> According to the World Health Organization (WHO), the current global life expectancy at birth is approximately **73 years**. This estimate is based on data from the year 2019.

> However, it's important to note that life expectancy can vary widely by country, with some countries having much higher or lower life expectancies than the global average. In general, life expectancy tends to be higher in wealthier countries with better access to healthcare and other resources.

REFERENCES

https://www.who.int/data/gho/data/themes/mortality-and-global-health-estimates/ghe-life-expectancy-and-healthy-life-expectancy

---

> **LIFE EXPECTANCY : DATASET**

> Life Expectancy is the statistical measure of the average time a person is expected to live based on a variety of factors such as age, sex, health, demographic factors etc.

> The **WHO** Dataset consists **2938** rows and **22** columns

> The columns are as followed

1. **COUNTRY** : Country name
2. **YEAR**: Year ranges from 2000 - 2015
3. **LIFE EXPECTANCY :** Life Expectancy in age
4. **ADULT MORTALITY :** Probability of dying between 15-60 years per 1000 population
5. **INFANT DEATHS :** Number of infant deaths per 1000 population
6. **ALCOHOL :** Recorded per capita consumption in litres
7. **PERCENTAGE :** Recorded as percentage of Gross Product per capita(%)
8. **HEPATITIS B:** Immunization coverage among 1 year old
9. **MEASLES :** Number of cases reported per 1000 population in percentage (%)
10. **BMI :** Average Body Mass Index of the population

11. **UNDER-FIVE-DEATHS:** Number of under five deaths per 1000 population
12. **POLIO :** Immunization coverage among 1 year old in percentage (%)
13. **TOTAL EXPENDITURE :** Government expenditure on heath as a percentage of total government expenditure
14. **DIPTHERIA :** DPT Immunization among the 1 year old in percentage (%)
15. **HIV/AIDS :** Deaths per 1000 live births
16. **POPULATION :** Population of the country
17. **THINNESS 1-19 YEARS:** Prevalence of thinness among children and adolescents for Age 10 to 19 (%)
18. **INCOME COMPOSITION OF RESOURCES:** Human Development Index based on income and availability of resources. Ranges between 0 and 1
19. **THINNESS 5-9 YEARS :** Prevalence of thinness among children and adolescents for Age 5 to 9 (%)
20. **SCHOOLING :** Number of years of Schooling in years
21. **GDP :** Gross Domestic Product per vcapita (in USD)
22. **STATUS** : Developed or Developing Country

- There are only 2 **CATEGORICAL** variables : **COUNTRY, STATUS**
- All other column values are **NUMERIC** VARIABLES

UNDERSTANDING LINEAR REGRESSION

> In today's data-driven business environment, retrieving and analyzing data has become critical to derive valuable insights and meet business requirements. However, the mere collection of data does not add any value to the organization. Historical data often plays a vital role in determining future goals, such as predicting a person's cholesterol level based on past medical records or determining their ability to repay a loan based on their credit history.

> In this project, I analyzed and predicted `life expectancy` based on various factors, including `adult mortality rates`, the `GDP` of the country, and other demographic indicators. Predicting the value of a specific variable based on other variables is the essence of Linear Regression analysis.

> Linear Regression problems can be categorized into two types:
>
> 1. Simple Linear Regression
> 2. Multiple Linear Regression.

---

1. ***Simple Linear Regression***

> focuses on utilizing only one variable `(X)` to predict the target variable `(Y)`. Here, `X` is known as an independent variable, and `Y` is the dependent or response variable.
>
> **For instance:** ***to predict life expectancy based only on a country's GDP***, the `GDP` would be the independent variable `(X)`, and `life expectancy` would be the dependent variable `(Y)`.

---

2. ***Multiple Linear Regression***

> In this project we use Multiple Linear Regression to predict the life expectancy focusing on utilizing more than one variable `(X1, X2, X3..)` to predict the target variable `(Y)`. Here, `(X1, X2, X3 ..)` is known as an independent variable, and `Y` is the dependent or response variable.
>
> **For instance:** ***to predict life expectancy based only on a country's GDP, adult mortality, income of people***, the `GDP`, `adult mortality`, `income of people` would be the independent variable `(X1, X2, X3)`, and `life expectancy` would be the dependent variable `(Y)`.

> To predict the target variable in life expectancy given the GDP and other predictor variables, we aim to estimate a function that provides the best possible prediction. This function can be represented by the following formula:

$$\hat{Y} = \hat{f}(X)$$

> $\hat{f}(X)$ **= Predicted function/model**
>
> The model is trained on a set of predictor variables `[X]` and response variables `[Y]` using a training dataset. The resulting function or model represents the learned relationship between the inputs `[X]` and outputs `[Y]` based on the training data.
>
> (X) **= Pedictor variables, also known as features or Independant variables**
>
> $\hat{Y}$ **= Predicted value of the output variable**

Statistical Formula: Simple Linear Regression

$$Y = \beta 0 + \beta 1 X + \epsilon$$

> Y *= the dependent variable*
>
> X *= the independent variable*
>
> $\beta 0$ *= the intercept*
>
> $\beta 1$ *= the slope coefficient*
>
> ϵ *= the error term or residual*

Statistical Formula: Simple Linear Regression for Predicting `Life Expectancy` based on `GDP`

$$Y = \beta 0 + \beta 1 (GDP) + \epsilon$$

Statistical Formula: Multiple Linear Regression

$$Y = \beta 0 + \beta 1 X1 + \beta 2 X2 + + \beta 3 X3 + \ldots + \beta n Xn + \epsilon$$

> Y *= the dependent variable*
>
> X *= the independent variable*
>
> $\beta 0$ *= the intercept*

$\beta 1, \beta 2X2, \beta 3X3....\beta nXn$ = *the slope coefficient of independant variables*

ϵ = *the error term or residual*

Statistical Formula: Multiple Linear Regression for Predicting `Life Expectancy` ***based on*** `GDP`, `adult_mortality`, `income`.... `n` ***independant variables***

$$Y = \beta 0 + \beta 1(GDP) + \beta 2(adultmortality) + \beta 3(income) + . .$$

CONCLUSION

In conclusion, the ability to analyze and predict outcomes based on historical data is crucial for making informed decisions in today's business environment. Linear Regression is a powerful tool for analyzing relationships between variables and making predictions, and it is important to understand the difference between Simple and Multiple Linear Regression in order to apply it effectively in real-world scenarios. Regenerate response

---

IMPORTANT METRICS

---

1. **MSE (Mean Squared Error)** is a way to measure how well a model predicts values by calculating the average of the squares of the differences between the predicted values and the actual values. It puts more emphasis on larger differences.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

---

2. **RMSE (Root Mean Squared Error)** is a measure of how well a model predicts continuous values, and is calculated by taking the square root of the MSE. The lower the RMSE, the better the model's performance.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

---

3. **MAE (Mean Absolute Error)** is another measure of how well a model predicts values, calculated by taking the average of the absolute differences between predicted and actual values. The smaller the MAE, the better the model's performance.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$$

---

4. **RMSLE (Root Mean Squared Logarithmic Error)** is a variation of RMSE, which is used when under-prediction is worse than over-prediction or when large differences in predictions and actual values need to be accounted for. It measures the ratio between actual values and predicted values, taking the logarithm of both.

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(1 + y_i) - \log(1 + \hat{y_i}))^2}$$

---

5. **R^2 (R Squared)** is a measure of how well a model's predicted values match the actual values. It ranges from 0 to 1, with 0 indicating no correlation and 1 indicating perfect correlation.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

---

6. **Mean Residual Deviance** measures how well the response variable is predicted by a model, taking into account the predictors included.

$$\text{Mean Residual Deviance} = \frac{1}{n-p}\sum_{i=1}^{n} d_i^2$$

---

7. **Null degrees of freedom** is the maximum number of independent values.

$$\boxed{\text{Null degrees of freedom} = n - 1}$$

In statistics, n usually refers to the sample size, which is the number of observations or data points in a sample. For example, if you have collected data on the height of 100 individuals, then n would be 100. The value of n is an important parameter in statistical analysis as it affects the precision and reliability of the estimates and inferences made from the data.

---

8. **Residual degrees of freedom** is the number of dimensions in which the residual vectors can vary, when fitting statistical models to data.

$$\boxed{\text{Residual degrees of freedom} = n - p - 1}$$

where n is the sample size and p is the number of predictors in the model. This will display the formula for Residual degrees of freedom inside a box. You can copy and paste this into a markdown cell in your Google Colab notebook.

---

9. **Null deviance** is a measure of how well a model predicts the response variable when only the intercept is included.

$$\boxed{\text{Null deviance} = -2\log(L_0)}$$

where L_0 is the likelihood of the null model, which is a model with only an intercept term (i.e., no predictors). The null deviance is a measure of the goodness of fit of the null model to the data.

---

10. **Residual deviance** is a measure of the goodness of a model's fit, with smaller values indicating better fit.

$$\boxed{\text{Residual deviance} = -2\log(L)}$$

where L is the likelihood of the model after fitting to the data. The residual deviance is a measure of the goodness of fit of the model after accounting for the predictors.

---

11. **AIC (Akaike Information Criterion)** is a measure of the quality of a model, taking into account the number of predictors included. More complex models are penalized to prevent irrelevant predictors from being included. The lower the AIC, the better the model.

$$\boxed{\text{AIC} = -2\log(L) + 2p}$$

where L is the likelihood of the model after fitting to the data, and p is the number of model parameters. The AIC is a measure of the relative quality of a statistical model for a given set of data, with lower values indicating a better model fit.

INSTALLATIONS

```
!pip install Jinja2==3.0.0
!pip install eli5==0.12.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting Jinja2==3.0.0
  Downloading Jinja2-3.0.0-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.4/133.4 kB 2.9 MB/s eta 0:00:0
Requirement already satisfied: MarkupSafe>=2.0.0rc2 in /usr/local/lib/python3.9/
Installing collected packages: Jinja2
  Attempting uninstall: Jinja2
    Found existing installation: Jinja2 3.1.2
    Uninstalling Jinja2-3.1.2:
      Successfully uninstalled Jinja2-3.1.2
Successfully installed Jinja2-3.0.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting eli5==0.12.0
  Downloading eli5-0.12.0.tar.gz (216 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.1/216.1 kB 4.5 MB/s eta 0:00:0
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.9/di
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-package
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: MarkupSafe>=2.0.0rc2 in /usr/local/lib/python3.9/
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-pa
```

```
  Building wheels for collected packages: eli5
    Building wheel for eli5 (setup.py) ... done
    Created wheel for eli5: filename=eli5-0.12.0-py2.py3-none-any.whl size=107622
    Stored in directory: /root/.cache/pip/wheels/c8/4d/e1/62d431c02b9c5e696f4bb5ca
  Successfully built eli5
  Installing collected packages: eli5
  Successfully installed eli5-0.12.0
```

```
!pip install h2o
#create data report
!pip install dataprep
```

```
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: webencodings in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packag
Building wheels for collected packages: metaphone
  Building wheel for metaphone (setup.py) ... done
  Created wheel for metaphone: filename=Metaphone-0.6-py3-none-any.whl size=1391
  Stored in directory: /root/.cache/pip/wheels/b2/9e/d9/26be7687b8fe36cd6cacbec3
Successfully built metaphone
Installing collected packages: regex, python-stdnum, python-crfsuite, pure_eval,
  Attempting uninstall: regex
    Found existing installation: regex 2022.10.31
    Uninstalling regex-2022.10.31:
      Successfully uninstalled regex-2022.10.31
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 2.0.9
    Uninstalling SQLAlchemy-2.0.9:
      Successfully uninstalled SQLAlchemy-2.0.9
Successfully installed aiohttp-3.8.4 aiosignal-1.3.1 asttokens-2.2.1 async-timed
```

```
!pip install shap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting shap
  Downloading shap-0.41.0-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.w
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 572.4/572.4 kB 10.1 MB/s eta 0:00:00
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: numba in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.9/dist-pack
Collecting slicer==0.0.7
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/pyth
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-package
Installing collected packages: slicer, shap
Successfully installed shap-0.41.0 slicer-0.0.7
```

IMPORT LIBRARIES

```
import h2o
from dataprep.eda import create_report
from h2o.automl import H2OAutoML
from jinja2.filters import do_default
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from google.colab import files
from scipy.stats.mstats import winsorize
from statsmodels.graphics.gofplots import qqplot
from sklearn.preprocessing import normalize, RobustScaler
import io
from sklearn.model_selection import train_test_split

from sklearn.feature_selection import f_regression
import pandas as pd
import statsmodels.api as sm
from pandas.testing import assert_frame_equal

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import datasets, linear_model

from sklearn import metrics
import eli5
from eli5.sklearn import PermutationImportance

from sklearn.impute import SimpleImputer
import statistics

from jinja2.filters import do_default

from sklearn.feature_selection import f_regression
import statsmodels.api as sm
from pandas.testing import assert_frame_equal
from sklearn.feature_selection import SelectKBest, mutual_info_regression
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor, plot_tree
```

```
import shap
```

> **FUNCTIONS**

Function 1 : cleanColumnNames

> Function defined to clean column names

```
def cleanColumnNames(df):
  #remove trailing and leading spaces
  df=df.rename(columns=lambda x :  x.strip())

  #remove extra spaces
  df.columns = df.columns.str.replace('  ',' ')

  #replace space with underscore
  df.columns=df.columns.str.replace(' ','_')

  #lowercase the column names
  df = df.rename(columns=lambda x: x.lower())

  return df
```

Function 2 : null_information

> Function defined to obtain the null information the columns of a dataframe

```
#Find null stats

def null_information(df):
  df_columns = list(df.columns)

  for colname in df_columns:
    null_rows = df[colname].isnull().sum()
    total_rows = df[colname].count()
    percentage_nulls = round((null_rows/total_rows)*100,2)
    # print("\ncolumn name --> ", colname)
```

```
    # print( "null rows --> ", null_rows)
    # print( "total row count --> ", total_rows)
    # print( "percentage of null rows --> ", percentage_nulls,"")
    print(f"column : {colname} has {null_rows} null values ---> {percentage_nulls} %
```

Function 3 : get_percent_missing

> This function is used to find the percentage of null values in the respective columns

```
#get percentage of missing null values
def get_percent_missing(dataframe):

    percent_missing = round(dataframe.isnull().sum() * 100 / len(dataframe),2)
    missing_value_df = pd.DataFrame({'column_name': dataframe.columns,
                                     'percent_missing': percent_missing})
    return missing_value_df
```

Function 4 : fill_na

> This function is used to fill **1%, 5%, 10%** null values in the respective columns

```
#function to add Nan/ null values
def fill_na(df,feature):
  print(feature)
  df_impute = pd.DataFrame(df[f'{feature}'])

  df_impute[f'{feature}_percent_1'] = df[f'{feature}']
  df_impute[f'{feature}_percent_5'] = df[f'{feature}']
  df_impute[f'{feature}_percent_10'] = df[f'{feature}']
  display(df_impute)

  #NULL VALUES BEFORE ADDING NaN
  print("PERCENTAGE OF NULL VALUES BEFORE ADDING Nan\n")
  df = get_percent_missing(df_impute)
  display(df)

  #ADDING 1 % Nan VALUES TO THE COLUMNS
  df_impute.loc[df_impute.sample(frac=0.01).index, f'{feature}_percent_1'] = pd.np.na

  #ADDING 5 % Nan VALUES TO THE COLUMNS
  df_impute.loc[df_impute.sample(frac=0.05).index, f'{feature}_percent_5'] = pd.np.na
```

```
  #ADDING 10 % Nan VALUES TO THE COLUMNS
  df_impute.loc[df_impute.sample(frac=0.1).index, f'{feature}_percent_10'] = pd.np.na

  #NULL VALUES AFTER ADDING 1%, 5%, 10% NaN
  print("PERCENTAGE OF NULL VALUES AFTER ADDING Nan\n")
  df1 = get_percent_missing(df_impute)
  display(df1)

  return df_impute
```

Function 5 : impute_values

> This function is used to perform imputation for the null values present in the respective columns

> Types of imputation used:

- **MEAN**

> Mean imputation (MI) is one such method in which the mean of the observed values for each variable is computed and the missing values for that variable are imputed by this mean.

$$\boxed{\text{mean} = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

- **MEDIAN**

> replacing all occurrences of missing values (NA) within a variable by the mean (if the variable has a Gaussian distribution) or median (if the variable has a skewed distribution).

$$\text{median} = \begin{cases} x_{\frac{n}{2}} & \text{if } n \text{ is even} \\ \frac{x_{\frac{n+1}{2}} + x_{\frac{n-1}{2}}}{2} & \text{if } n \text{ is odd} \end{cases}$$

where n is the sample size and x_i is the value of the i-th observation. The median is the value that separates the higher half and the lower half of the dataset. If the sample size n is even, then the median is the average of the two middle values.

- **MOST FREQUENT**

This technique says to replace the missing value with the variable with the highest frequency or in simple words replacing the values with the Mode of that column. This technique is also referred to as Mode Imputation.

$$\text{most frequent} = \text{mode}(x_1, x_2, \ldots, x_n)$$

where n is the sample size and x_i is the value of the i-th observation. The most frequent value, or mode, is the value that appears most often in the dataset.

```
#function for imputation
from sklearn.impute import SimpleImputer

def impute_values(df, impute_method):
  cols = list(df.columns)
  #MEAN
  #cols.remove('winz_life_expectancy')
  if impute_method == 'mean':
    impute_mean = SimpleImputer(strategy=impute_method)
    impute_mean.fit(df)
    df_impute = impute_mean.transform(df)
    df_impute = pd.DataFrame(df_impute, columns = cols)

    # #NULL VALUES AFTER ADDING 1%, 5%, 10% NaN
    # print("PERCENTAGE OF NULL VALUES BEFORE ADDING Nan\n")
    # df1 = get_percent_missing(df_impute)
    # display(df1)

  #MEDIAN
  #cols.remove('winz_life_expectancy')
  elif impute_method == 'median':
    impute_mean = SimpleImputer(strategy=impute_method)
    impute_mean.fit(df)
```

```
    df_impute = impute_mean.transform(df)
    df_impute = pd.DataFrame(df_impute, columns = cols)

  #MOST_FREQUENT
  elif impute_method == 'most_frequent':
    impute_mean = SimpleImputer(strategy=impute_method)
    impute_mean.fit(df)
    df_impute = impute_mean.transform(df)
    df_impute = pd.DataFrame(df_impute, columns = cols)


    #NULL VALUES AFTER ADDING 1%, 5%, 10% NaN
    print("PERCENTAGE OF NULL VALUES BEFORE ADDING Nan\n")
    df1 = get_percent_missing(df_impute)
    display(df1)

  return df_impute
```

Function 6 : statistical_measures

> One of the primary objectives of this project is to implement a function that calculates the mathematical formulas for the statistical factors, namely **MSE** and **VARIANCE**.
>
> By doing so, we aim to gain a better understanding of the underlying mathematical concepts and to compare the results with the inbuilt functions provided by the sklearn library. Ultimately, our goal is to verify the accuracy and reliability of our function in calculating these important statistical factors.

> 1. **MSE (Mean Squared Error)** is a way to measure how well a model predicts values by calculating the average of the squares of the differences between the predicted values and the actual values. It puts more emphasis on larger differences.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

> where n is the number of samples in the dataset, y_i is the true value of the dependent variable for the i-th sample, and $\hat{y_i}$ is the predicted value of the

dependent variable for the i-th sample.

IMPLEMENTING THE ABOVE MSE FORMULA IN THE CODE

```
#FINDING MSE USING THE MATHEMATICAL FORMULA FOR 1% IMPUTATION
mse_formula = (pow((df['winz_life_expectancy'] - df['winz_life_expectancy_percent_1']),2))
print("MSE USING MATHEMATICAL FORMULA --> ", mse_formula )
```

In linear regression, the goal is to find a linear relationship between the dependent variable and one or more independent variables. The coefficients of the linear regression model are estimated by minimizing the MSE between the predicted values of the dependent variable and the true values in the dataset.

Thus, MSE is used to evaluate how well the linear regression model fits the data. ***A lower MSE indicates a better fit between the predicted values and the true values, while a higher MSE indicates a poorer fit.*** Therefore, minimizing the MSE is a common objective when fitting linear regression models.

---

2. **VARIANCE**

$$\boxed{\text{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

where n is the sample size, x_i is the value of the i-th observation, and μ is the sample mean. The variance measures how much the values in the dataset vary from the mean.

IMPLEMENTING THE ABOVE VARIANCE FORMULA IN THE CODE

```
#FINDING VARIANCE USING THE MATHEMATICAL FORMULA FOR 5% IMPUTATION
var_formula = round((pow((df['residual_percent_5'] - residual_mean_5),2)).sum()/length,5)
print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )
```

> In linear regression, the goal is to minimize the residual variance by selecting the best predictors and estimating their coefficients. A lower residual variance indicates a better fit of the model to the data. The residual variance is also used to calculate the standard error of the regression coefficients, which is a measure of the uncertainty in the estimated coefficients.

> Overall, variance is an important statistical concept in linear regression and is used to evaluate the quality of the model fit and estimate the uncertainty in the regression coefficients.

> I have used and implemented the mathematical formula to verify it with the `sklearn` inbuilt function.

```
#function for imputation
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_squared_error
import statistics

def statistical_measures(df):
  #RESIDUAL/ERROR = EXPECTED VALUE - PREDICTED VALUE
  length = df.shape[0]
  print("MATHEMATICAL FORMULA FOR FINDING MSE:")
  print('''
╔═══════════════════════════════════════════════╗
║  MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2          ║
╚═══════════════════════════════════════════════╝
''')
  print("MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:")
  print('''
╔═══════════════════════════════════════════════════════════╗
║  Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2  ║
╚═══════════════════════════════════════════════════════════╝
''')




  #FINDING MSE FOR 1% IMPUTATION
  df['residual_percent_1'] = df['winz_life_expectancy'] - df['winz_life_expectancy_pe
  df['residual_square_percent_1'] = pow(df['winz_life_expectancy'] - df['winz_life_ex
  df['residual_percentage_error_1'] = ((df['winz_life_expectancy'] - df['winz_life_ex
  residual_mean = df['residual_percent_1'].mean()
```

```
print("\n================================================================================
print("1% IMPUTATION")
print("================================================================================
print("\nMEAN SQUARE ERROR - MSE")
print("--------------------------------------------------------------------------------

#FINDING MSE USING THE MATHEMATICAL FORMULA FOR 1% IMPUTATION
mse_formula = (pow((df['winz_life_expectancy'] - df['winz_life_expectancy_percent_1
print("MSE USING MATHEMATICAL FORMULA --> ", mse_formula )

#FINDING MSE USING THE SCIKIT INBUILT FORMULA FOR 1% IMPUTATION
mse = mean_squared_error(df['winz_life_expectancy'] , df['winz_life_expectancy_perc
print("MSE USING SCIKIT FUNCTION --> ", mse )

print("\nVARIANCE")
print("--------------------------------------------------------------------------------
print("Mean value of the residuals --> ", residual_mean)

#FINDING VARIANCE USING THE MATHEMATICAL FORMULA FOR 1% IMPUTATION
var_formula = round((pow((df['residual_percent_1'] - residual_mean),2)).sum()/lengt
print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )

#FINDING VARIANCE USING THE SCIKIT INBUILT FORMULA FOR 1% IMPUTATION
var = round(statistics.variance(list(df['residual_percent_1'])),5)
print("VARIANCE USING SCIKIT FUNCTION --> ", var)

#FINDING MSE FOR 5% IMPUTATION
df['residual_percent_5'] = df['winz_life_expectancy'] - df['winz_life_expectancy_pe
df['residual_square_percent_5'] = pow(df['winz_life_expectancy'] - df['winz_life_ex
df['residual_percentage_error_5'] = ((df['winz_life_expectancy'] - df['winz_life_ex
residual_mean_5 = df['residual_percent_5'].mean()

print("\n================================================================================
print("5% IMPUTATION")
print("================================================================================
print("\nMEAN SQUARE ERROR - MSE")
print("--------------------------------------------------------------------------------

#FINDING MSE USING THE MATHEMATICAL FORMULA FOR 5% IMPUTATION
mse_formula = (pow((df['winz_life_expectancy'] - df['winz_life_expectancy_percent_5
print("MSE USING MATHEMATICAL FORMULA --> ", mse_formula )

#FINDING MSE USING THE SCIKIT INBUILT FORMULA FOR 5% IMPUTATION
mse = mean_squared_error(df['winz_life_expectancy'] , df['winz_life_expectancy_perc
print("MSE USING SCIKIT FUNCTION --> ", mse )

print("\nVARIANCE")
print("--------------------------------------------------------------------------------
```

```
print("Mean value of the residuals --> ", residual_mean_5)

#FINDING VARIANCE USING THE MATHEMATICAL FORMULA FOR 5% IMPUTATION
var_formula = round((pow((df['residual_percent_5'] - residual_mean_5),2)).sum()/len
print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )

#FINDING VARIANCE USING THE SCIKIT INBUILT FORMULA FOR 5% IMPUTATION
var = round(statistics.variance(list(df['residual_percent_5'])),5)
print("VARIANCE USING SCIKIT FUNCTION --> ", var)


#FINDING MSE FOR 10% IMPUTATION
df['residual_percent_10'] = df['winz_life_expectancy'] - df['winz_life_expectancy_p
df['residual_square_percent_10'] = pow(df['winz_life_expectancy'] - df['winz_life_e
df['residual_percentage_error_10'] = ((df['winz_life_expectancy'] - df['winz_life_e
residual_mean_10 = df['residual_percent_10'].mean()

print("\n=========================================================================
print("10% IMPUTATION")
print("=========================================================================
print("\nMEAN SQUARE ERROR - MSE")
print("-------------------------------------------------------------------------


#FINDING MSE USING THE MATHEMATICAL FORMULA FOR 10% IMPUTATION
mse_formula = (pow((df['winz_life_expectancy'] - df['winz_life_expectancy_percent_1
print("MSE USING MATHEMATICAL FORMULA --> ", mse_formula )

#FINDING MSE USING THE SCIKIT INBUILT FORMULA FOR 10% IMPUTATION
mse = mean_squared_error(df['winz_life_expectancy'] , df['winz_life_expectancy_perc
print("MSE USING SCIKIT FUNCTION --> ", mse )

print("\nVARIANCE")
print("-------------------------------------------------------------------------
print("Mean value of the residuals --> ", residual_mean_10)

#FINDING VARIANCE USING THE MATHEMATICAL FORMULA FOR 10% IMPUTATION
var_formula = round((pow((df['residual_percent_10'] - residual_mean_10),2)).sum()/l
print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )

#FINDING VARIANCE USING THE SCIKIT INBUILT FORMULA FOR 10% IMPUTATION
var = round(statistics.variance(list(df['residual_percent_10'])),4)
print("VARIANCE USING SCIKIT FUNCTION --> ", var)


return df
```

```
def calc_residual(df):
  df['residual_error'] =  df['life_expectancy'] - df[f'{col}']
  df['residuals_square'] = round(pow(df['residuals'] ,2),3)

  #SCATTER PLOT OF THE REAL VALUES VERSUS THE IMPUTED VALUES
  print("SCATTER PLOT OF THE REAL VALUES VERSUS THE IMPUTED VALUES")
  plt.scatter(df['life_expectancy'] , df[f'{col}'])
  plt.xlabel('life_expectancy [True Values]')
  plt.ylabel('life_expectancy Imputed Values')

  #RESIDUALS
  #RESIDUAL PLOT OF THE REAL VALUES VERSUS THE IMPUTED VALUES
  sns.distplot((df['life_expectancy']-df[f'{col}']), bins=50, color='purple')


  residual_variance = df['residuals_square'].sum()
  return df,residual_variance
```

Function 7 : residual_plot

> This function plots the residuals for all the imputation methods accross all 1%, 5% and 10% null values

```
from jinja2.filters import do_default
#SCATTER PLOT OF THE REAL TEST VALUES VERSUS THE PREDICTED VALUES

def residual_plot(df):
  plt.figure(figsize=(20,40))
  print("RESIDUAL PLOT FOR 1% IMPUTATION")
  plt.subplot(3,3,1)
  plt.scatter(df['winz_life_expectancy'], df['winz_life_expectancy_percent_1'])
  plt.xlabel('True Feature Values')
  plt.ylabel('Imputed Feature Values')

  p1 = max(max(df['winz_life_expectancy_percent_1']), max(df['winz_life_expectancy'])
  p2 = min(min(df['winz_life_expectancy_percent_1']), min(df['winz_life_expectancy'])
  plt.plot([p1, p2], [p1, p2], 'r-')

  plt.subplot(3,3,2)
  plt.scatter(df['winz_life_expectancy']-df['winz_life_expectancy_percent_1'],df['win
  plt.xlabel('Predicted Value')
  plt.ylabel('Residual Error')
  plt.title('PLOT FOR 1 % MEAN IMPUTATION ', fontsize = 16)

  plt.subplot(3,3,3)
```

```
  plt.hist(df['winz_life_expectancy']-df['winz_life_expectancy_percent_1'],bins=50, c
  plt.xlabel('Distribution Of Error')
  #plt.ylabel('Residual Error')

  print("RESIDUAL PLOT FOR 5% IMPUTATION")
  plt.subplot(3,3,4)
  plt.scatter(df['winz_life_expectancy'], df['winz_life_expectancy_percent_5'])
  plt.xlabel('True Feature Values')
  plt.ylabel('Imputed Feature Values')

  p1 = max(max(df['winz_life_expectancy_percent_5']), max(df['winz_life_expectancy'])
  p2 = min(min(df['winz_life_expectancy_percent_5']), min(df['winz_life_expectancy'])
  plt.plot([p1, p2], [p1, p2], 'r-')

  plt.subplot(3,3,5)
  plt.scatter(df['winz_life_expectancy']-df['winz_life_expectancy_percent_5'],df['win
  plt.xlabel('Predicted Value')
  plt.ylabel('Residual Error')
  plt.title('PLOT FOR 5 % MEAN IMPUTATION ', fontsize = 16)

  plt.subplot(3,3,6)
  plt.hist(df['winz_life_expectancy']-df['winz_life_expectancy_percent_5'],bins=50, c
  plt.xlabel('Distribution Of Error')
  #plt.ylabel('Residual Error')

  print("RESIDUAL PLOT FOR 10% IMPUTATION")
  plt.subplot(3,3,7)
  plt.scatter(df['winz_life_expectancy'], df['winz_life_expectancy_percent_10'])
  plt.xlabel('True Feature Values')
  plt.ylabel('Imputed Feature Values')

  p1 = max(max(df['winz_life_expectancy_percent_10']), max(df['winz_life_expectancy']
  p2 = min(min(df['winz_life_expectancy_percent_10']), min(df['winz_life_expectancy']
  plt.plot([p1, p2], [p1, p2], 'r-')

  plt.subplot(3,3,8)
  plt.scatter(df['winz_life_expectancy']-df['winz_life_expectancy_percent_10'],df['wi
  plt.xlabel('Predicted Value')
  plt.ylabel('Residual Error')
  plt.title('PLOT FOR 10 % MEAN IMPUTATION ', fontsize = 16)

  plt.subplot(3,3,9)
  plt.hist(df['winz_life_expectancy']-df['winz_life_expectancy_percent_10'],bins=50,
  plt.xlabel('Distribution Of Error')


#RESIDUALS
#plt.subplot(3,2,2)
#sns.displot((df['winz_life_expectancy']-df['winz_life_expectancy_percent_1']),kde=Tr
```

Function 8: partial_dependence_plot

> Function defined for PDP Analysis

```
def partial_dependence_plot(model,feature, idx=None):
    if idx is None: # visualize all samples
        shap.plots.partial_dependence(
            feature,
            model.predict,
            X_train,
            ice=False,
            model_expected_value=True,
            feature_expected_value=True)
    else: # visualize sample idx
        shap.partial_dependence_plot(
            feature,
            model.predict,
            X_train,
            ice=False,
            model_expected_value=True,
            feature_expected_value=True,
            shap_values=shap_values[idx:idx+1,:])
```

Function 9: prediction

> Function defined for Model Prediction

```
def prediction(X_train, X_test, y_train, y_test):
  # Predict on training and testing data
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate MAE on training and testing data
    train_mae = mean_absolute_error(y_train, y_train_pred)
    test_mae = mean_absolute_error(y_test, y_test_pred)

    print("Training MAE: ", train_mae)
    print("Testing MAE: ", test_mae)

    # Calculate the RMSE on the training and testing data
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
```

```
    print("\nTraining RMSE: ", train_rmse)
    print("Testing RMSE: ", test_rmse)
```

READ THE DATA

```
url = 'https://raw.githubusercontent.com/ShreyaJaiswal1604/Coursework-Data-Science-En
```

```
df_original = pd.read_csv(url)
```

```
df_original.head()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure
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 rows × 22 columns

CLEANING THE DATA COLUMNS NAMES

> Cleaning data column names in machine learning involves standardizing, shortening, and adding descriptive names to make them more informative and consistent. This improves organization and readability of the data, and helps to avoid confusion and errors during analysis.

> Here are the key points regarding cleaning data column names in machine learning

> - Standardize the naming convention for consistency throughout the dataset
> - Remove special characters and spaces to avoid errors and improve readability
> - Shorten long column names using abbreviations or acronyms while retaining meaning
> - Add descriptive names to column headers to make the data more informative
> - Cleaning data column names is an important step in machine learning
> - preprocessing, improving organization and avoiding confusion or errors during analysis.

```
#Using the cleanColumnName function to clean the column names
df_original = cleanColumnNames(df_original)
df_original.head()
```

	country	year	status	life_expectancy	adult_mortality	infant_deaths	al
0	Afghanistan	2015	Developing	65.0	263.0	62	
1	Afghanistan	2014	Developing	59.9	271.0	64	
2	Afghanistan	2013	Developing	59.9	268.0	66	
3	Afghanistan	2012	Developing	59.5	272.0	69	
4	Afghanistan	2011	Developing	59.2	275.0	71	

5 rows × 22 columns

```
#Displays statistical information of the numeric columns
df_original.describe()
```

	year	life_expectancy	adult_mortality	infant_deaths	alcohol	pe
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	
std	4.613841	9.523867	124.292079	117.926501	4.052413	
min	2000.000000	36.300000	1.000000	0.000000	0.010000	
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	

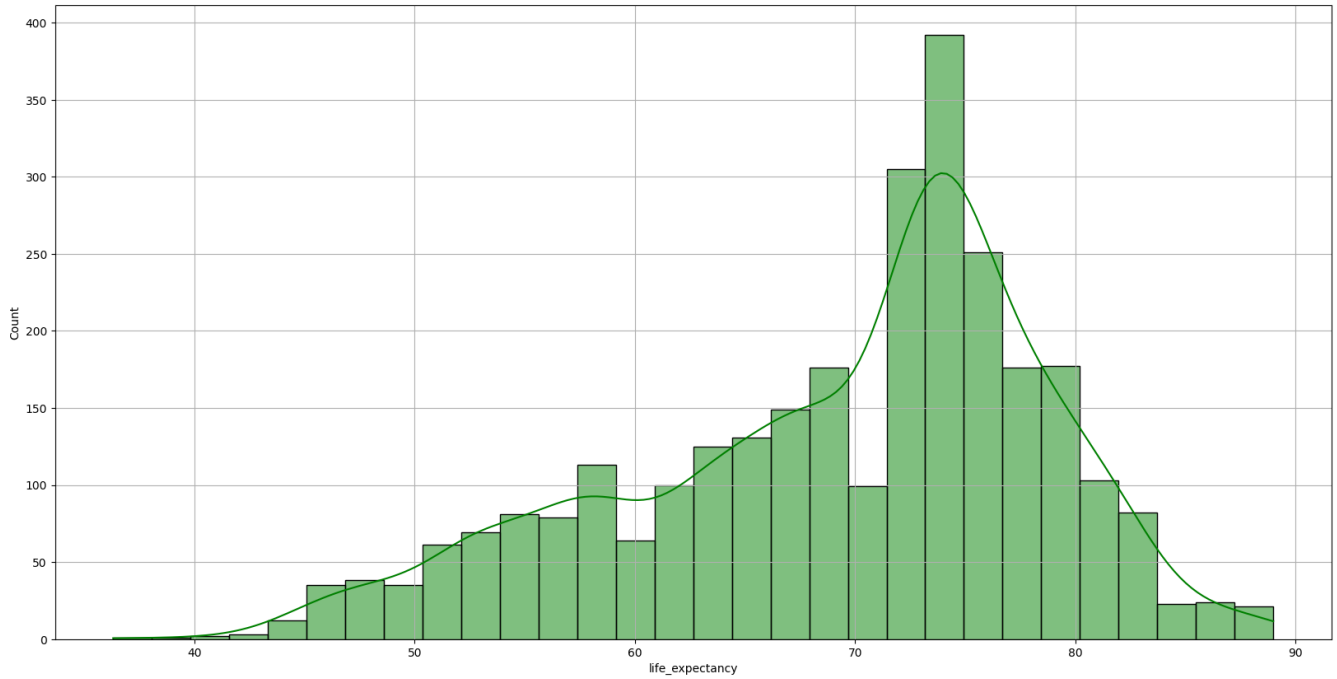
EXPLORATORY DATA ANALYSIS (EDA)

> Using Exploratory Data Analysis (EDA), we can analyze and investigate the data sets and summarize the required characteristics mainly through visualizations.

> PYTHON LIBRARIES USED:
>
> 1. Seaborn
> 2. Matplotlib
> 3. Plotly

Distribution of Life Expectancy

```
plt.figure(figsize=(20,10))
sns.histplot(df_original['life_expectancy'].dropna(), kde=True, color="green")
plt.grid(True)
```



Correlation Heatmap for dependency Visualization

HEATMAP

> A correlation heatmap is a heatmap that shows a 2D correlation matrix between two discrete dimensions, using colored cells to represent data from usually a monochromatic scale

> - The parameter cmap can takes various existing color mapping such as
>
> 1. mako
> 2. coolwarm
> 3. YlGnBu
> 4. Spectral
> 5. crest

> - You can define your own cmap palette as well

> **REFERENCE**
>
> https://seaborn.pydata.org/generated/seaborn.heatmap.html
> https://seaborn.pydata.org/examples/many_pairwise_correlations.html

---

CORRELATION MATRIX

1. Correlation defines how two or more variables are related to one another.
2. Correlation matrix is a table which displays the coefficients for the variables present in the table. It defines the extent of interdependency between the variables.
3. **-1** defines a prefectly negative correlation between the variables
4. **0** defines no correlation between the variables
5. **1** defines a perfectly positive correlation between the variables

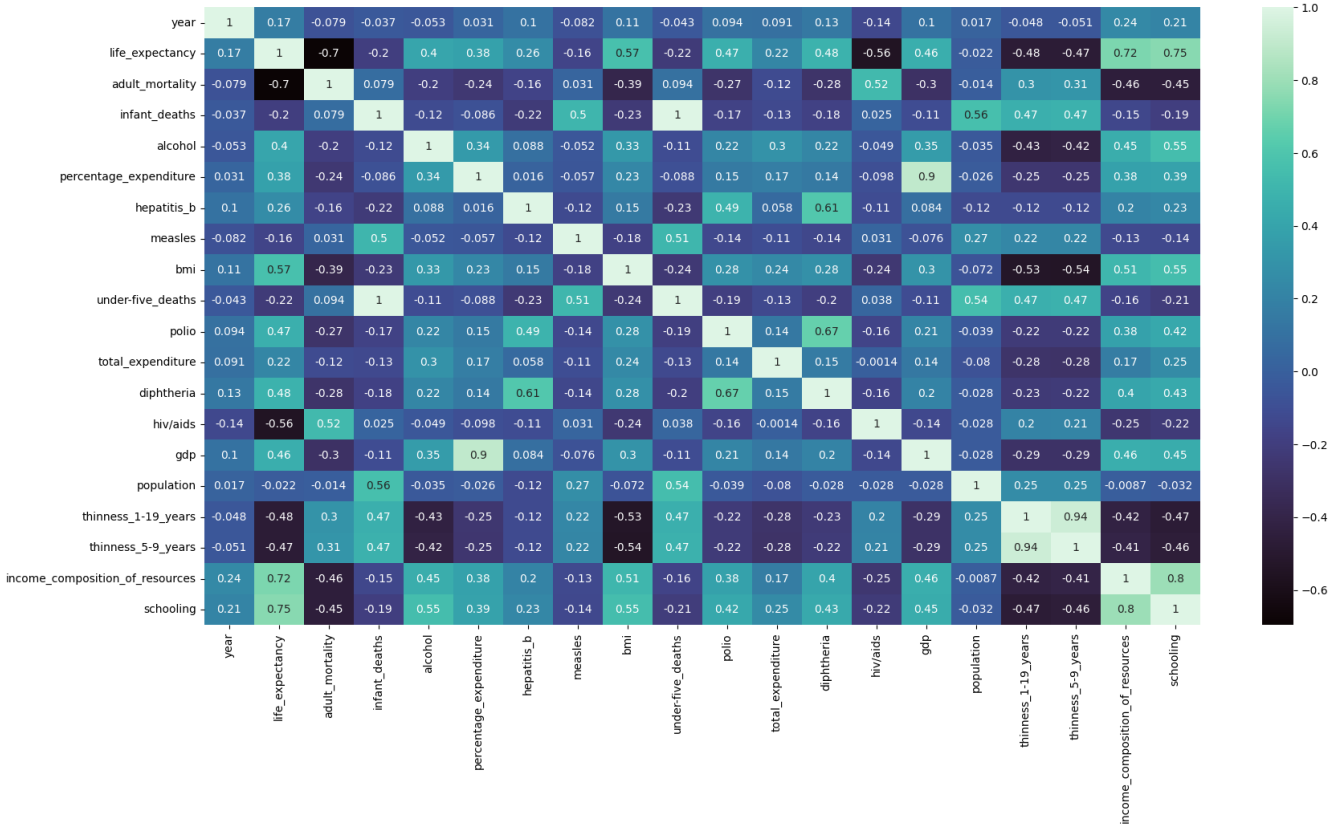
   REFERENCES

   https://www.statology.org/how-to-read-a-correlation-matrix/ https://muthu.co/understanding-correlations-and-correlation-matrix/ https://www.wallstreetmojo.com/correlation-matrix/#h-what-is-the-correlation-matrix

---

```
#PLOTTING THE HEATMAP
plt.figure(figsize=(20,10))
sns.heatmap(df_original.corr(), annot=True, cmap="mako")
```

```
The default value of numeric_only in DataFrame.corr is deprecated. In a future v
<Axes: >
```



INFERENCE

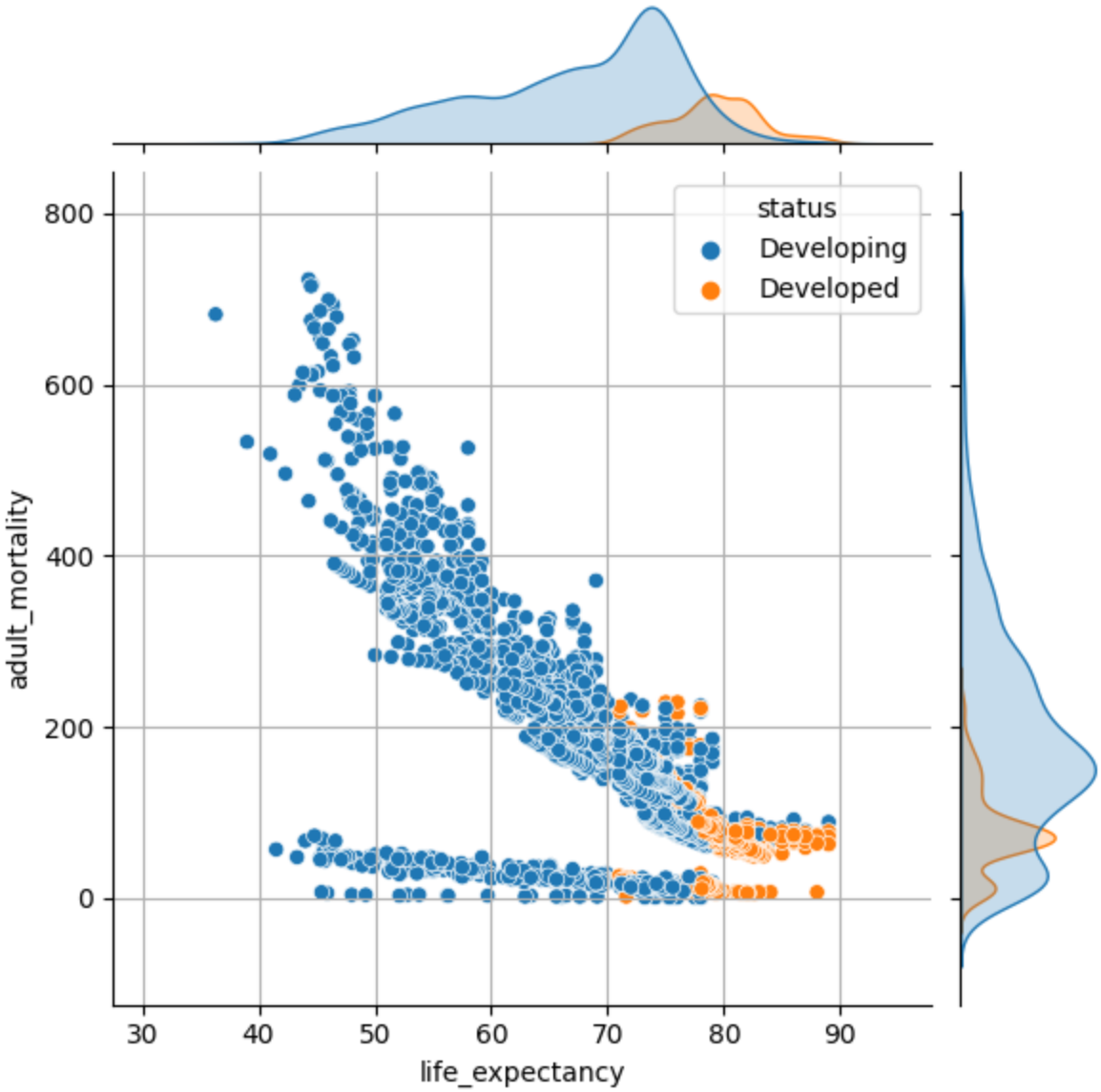
Insights obtained from the correlation matrix

1. Correlation value between Life Expectancy and Adult Mortality relate is **-0.7**.Therefore a high negative correlation is anticipated between them.
2. Correlation value between GDP and Life Expectancy is **0.46**. It exhibits a positive correlation, which can be inferred that as the GDP inceases, the life expectancy also increases.
3. BMI also has a positive correlation of **0.57**.
4. Schooling years also have positive correlation with Life Expectancy, with the value of **0.75**
5. Income composition of resources also exhibits a high correlation with Life Expectancy of **0.72**. Which can be inferred as more the income and availability of resources leads to high Life Expectancy.

Plotting Life Expectance vs Adult Mortality for developed and developing countries

```
plt.figure(figsize=(20,10))
sns.jointplot(data=df_original, y=df_original['adult_mortality'], x=df_original['life
plt.grid(True)
```

```
<Figure size 2000x1000 with 0 Axes>
```



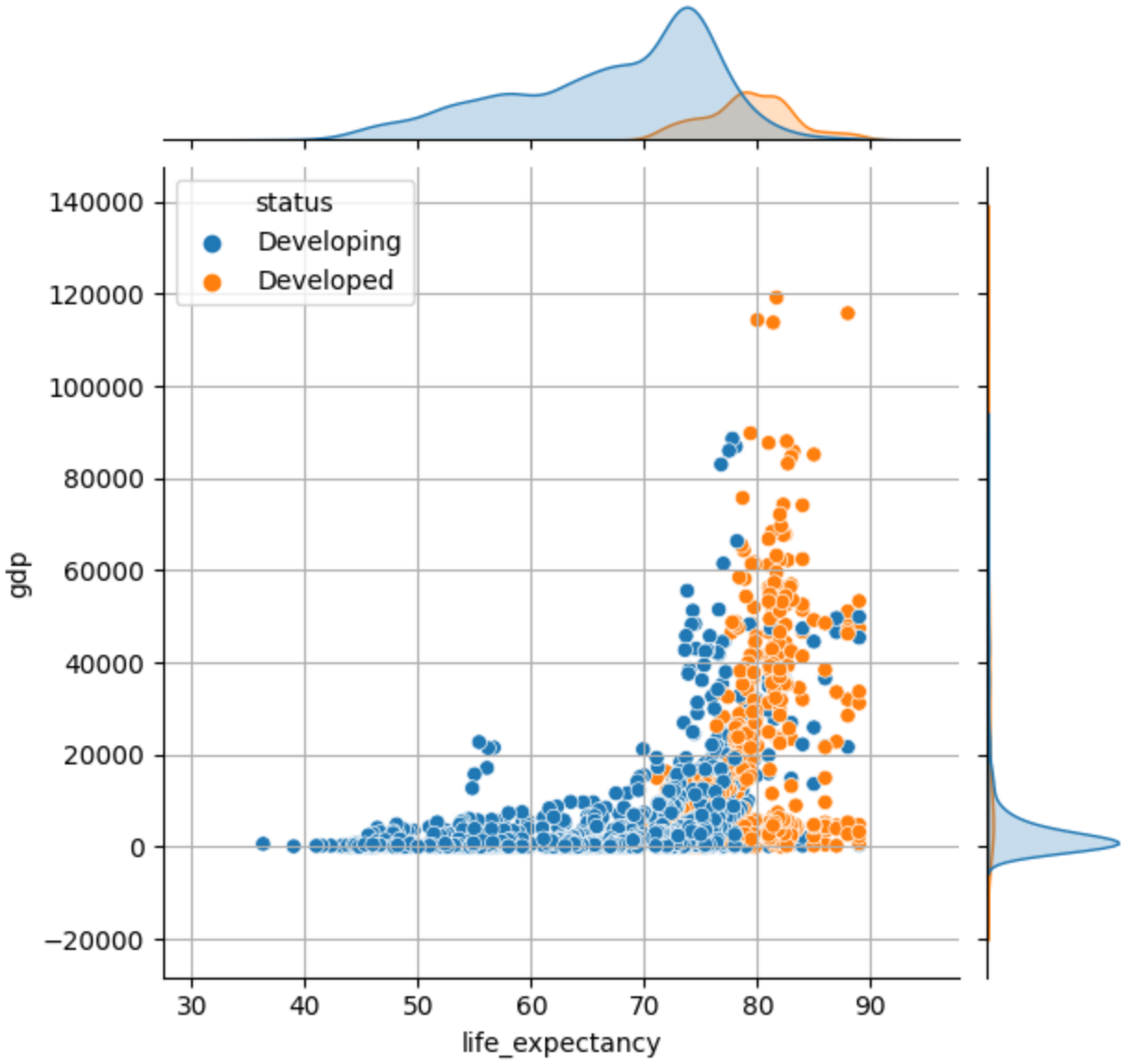
INFERENCE

> We see that the developed countries have low adult mortality which leads to high life expectancy

Plotting Life Expectance vs GDP for developed and developing countries

```
plt.figure(figsize=(20,10))
sns.jointplot(data=df_original, y=df_original['gdp'], x=df_original['life_expectancy'
plt.grid(True)
```

```
<Figure size 2000x1000 with 0 Axes>
```



INFERENCE

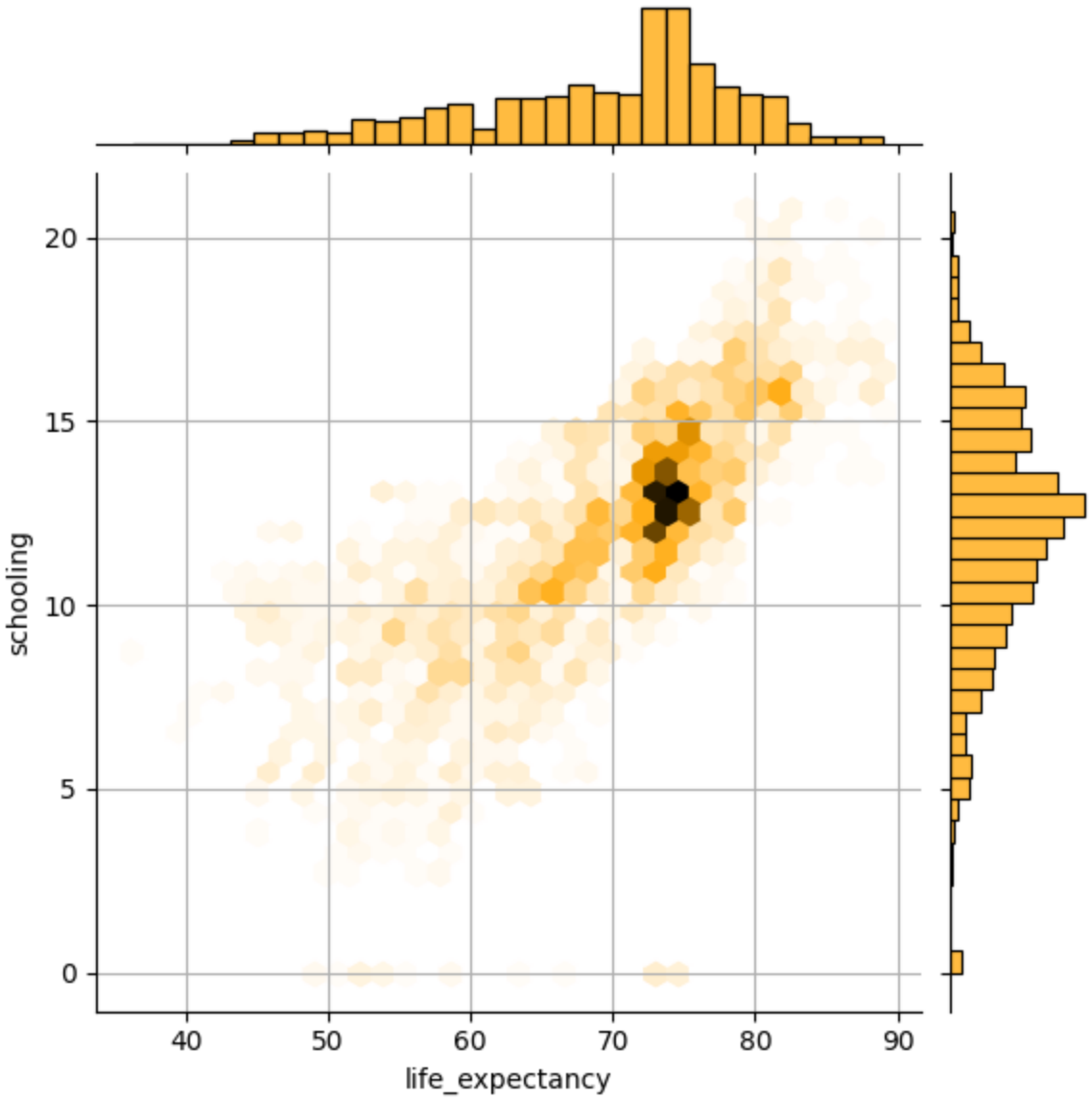
> We see that most of the developed countries have a high GDP leading to a higher life expectancy when compared to the developing countries

Plotting Life Expectance vs Schooling using hex plot

> hex plot represents a 2D histogram plot, in which the bins are hexagons and the color represents the number of data points within each bin.

```
plt.figure(figsize=(20,10))
sns.jointplot(data=df_original, y=df_original['schooling'], x=df_original['life_expec
plt.grid(True)
```

```
<Figure size 2000x1000 with 0 Axes>
```



INFERENCE

> As the year of schooling increases it is observed that the life expectancy also increases.

Plotting Life Expectance vs income composition of resources for developed and developing countries using KDE plot

---

> A kernel density estimate (KDE) plot is a method for visualizing the distribution of observations in a dataset, analogous to a histogram.

```
plt.figure(figsize=(20,10))
sns.jointplot(data=df_original, y=df_original['income_composition_of_resources'], x=d
plt.grid(True)
```

```
<Figure size 2000x1000 with 0 Axes>
```

INFERENCE

> We see that the devloped countries have a range towards the high values of both income composition of resources and life expectancy. This means that developed countries have more income and avaliability of resources which leads to a higher life expectancy.

PAIR PLOTS

> The Seaborn Pairplot allows us to plot pairwise relationships between variables within a dataset.

> https://seaborn.pydata.org/generated/seaborn.pairplot.html

```
plt.figure(figsize=(20,10))
sns.pairplot(df_original, palette=sns.color_palette("pastel"))
plt.grid(True)
```

```
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
Ignoring `palette` because no `hue` variable has been assigned.
```

```
Ignoring `palette` because no `hue` variable has been assigned.
```

DATA PREPROCESSING

- Data preprocessing is a data mining technique which transforms the data into an understandable format.
- Real-world data are not likely to be consistent and may have a lot of variations. It is therefore important to pre process the data before implementing any modeling technique to get a better and more accurate result.

```
Ignoring `palette` because no `hue` variable has been assigned.
```

CHECK THE NULL VALUES

```
Ignoring `palette` because no `hue` variable has been assigned.
```

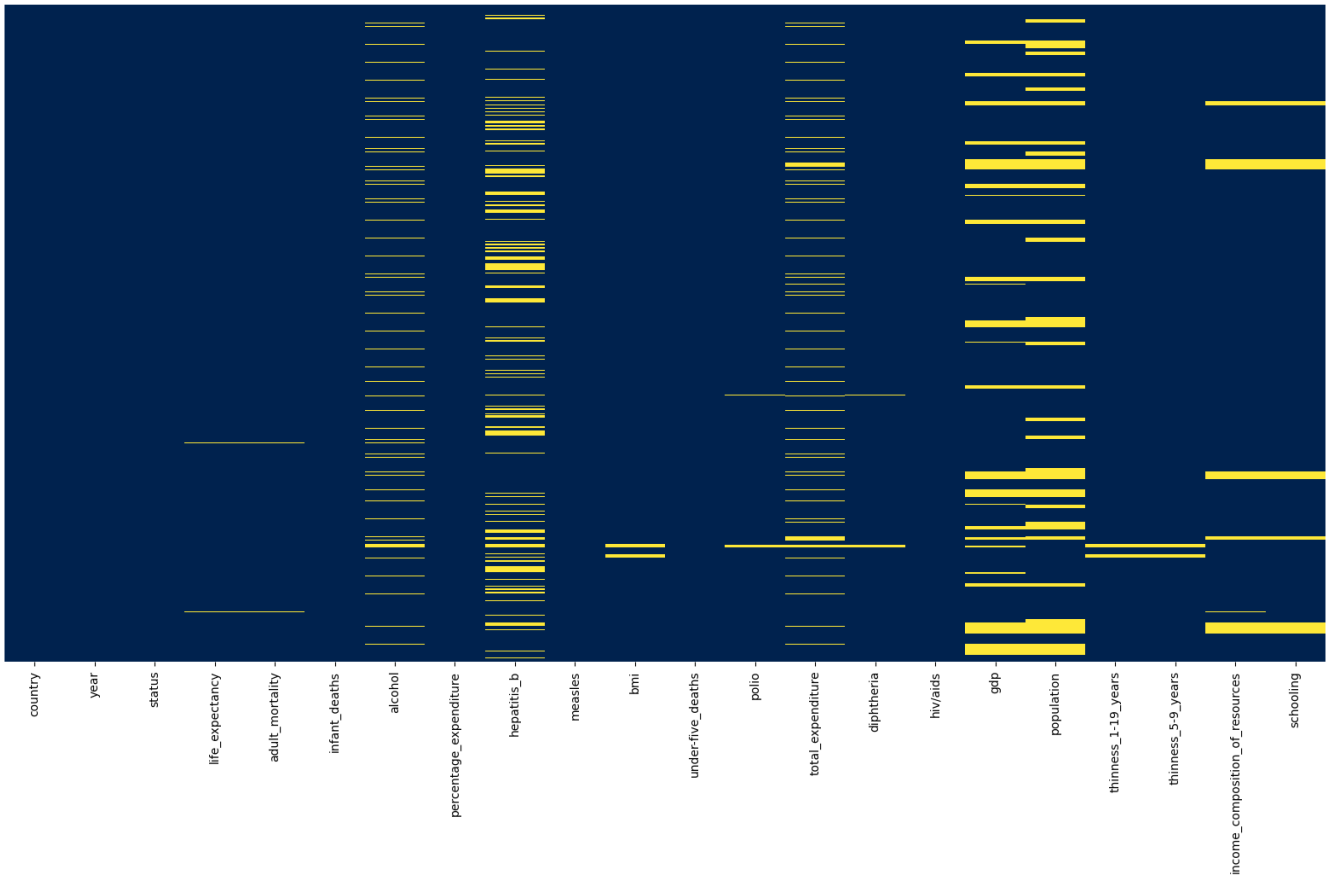
```
#checking null values
df_original.isnull().sum()
```

```
country                             0
year                                0
status                              0
life_expectancy                    10
adult_mortality                    10
infant_deaths                       0
alcohol                           194
percentage_expenditure              0
hepatitis_b                       553
measles                             0
bmi                                34
under-five_deaths                   0
polio                              19
total_expenditure                 226
diphtheria                         19
hiv/aids                            0
gdp                               448
population                        652
thinness_1-19_years                34
thinness_5-9_years                 34
income_composition_of_resources   167
schooling                         163
dtype: int64
```

```
Ignoring `palette` because no `hue` variable has been assigned.
```

```
#plotting heatmap to check the null values
plt.figure(figsize=(20,10))
sns.heatmap(df_original.isnull(),yticklabels=False,cbar=False,cmap='cividis')
```

```
<Axes: >
```




```
Ignoring `palette` because no `hue` variable has been assigned.
```

```
#getting null information
null_information(df_original)
```

```
column : country has 0 null values ---> 0.0 % of nulls
column : year has 0 null values ---> 0.0 % of nulls
column : status has 0 null values ---> 0.0 % of nulls
column : life_expectancy has 10 null values ---> 0.34 % of nulls
column : adult_mortality has 10 null values ---> 0.34 % of nulls
column : infant_deaths has 0 null values ---> 0.0 % of nulls
column : alcohol has 194 null values ---> 7.07 % of nulls
column : percentage_expenditure has 0 null values ---> 0.0 % of nulls
column : hepatitis_b has 553 null values ---> 23.19 % of nulls
column : measles has 0 null values ---> 0.0 % of nulls
column : bmi has 34 null values ---> 1.17 % of nulls
column : under-five_deaths has 0 null values ---> 0.0 % of nulls
column : polio has 19 null values ---> 0.65 % of nulls
column : total_expenditure has 226 null values ---> 8.33 % of nulls
column : diphtheria has 19 null values ---> 0.65 % of nulls
column : hiv/aids has 0 null values ---> 0.0 % of nulls
column : gdp has 448 null values ---> 17.99 % of nulls
column : population has 652 null values ---> 28.52 % of nulls
column : thinness_1-19_years has 34 null values ---> 1.17 % of nulls
column : thinness_5-9_years has 34 null values ---> 1.17 % of nulls
column : income_composition_of_resources has 167 null values ---> 6.03 % of null
column : schooling has 163 null values ---> 5.87 % of nulls
Ignoring `palette` because no `hue` variable has been assigned.
```

INFERENCE

From the above values, we can see that the following columns have null values

1. life_expectancy
2. adult_mortality
3. alcohol
4. hepatatis_b
5. bmi
6. polio
7. total_expenditure
8. diptheria
9. gdp
10. population
11. thinness_1-19_years
12. thinness_5-9_years
13. income_composition_of_resources
14. schooling

> There are many columns which have null values, however, the number of records are not large enough to drop the data. Dropping the data would further lead to the loss of some important information and degrade our model.

> Therefore, Imputation of these missing values would be a better option.

`ignoring "palette" because no "hue" variable has been assigned.`

> **IMPUTATION**

> Data imputation is a method for retaining the majority of the dataset's data and information by substituting missing data with a different value.

> Data imputation is a method used in data preprocessing to fill in missing or incomplete data values with estimated values. The process involves using statistical techniques to estimate the missing data points based on the available information in the dataset.

> There are various techniques used for data imputation, including:

> 1. Mean imputation: Filling in missing values with the mean value of the column.
> 2. Median imputation: Filling in missing values with the median value of the column.
> 3. Regression imputation: Using regression analysis to estimate missing values based on the relationship between the variables.
> 4. Hot deck imputation: Imputing missing values using information from similar cases in the dataset.
> 5. Multiple imputation: Generating multiple imputed datasets based on statistical models to provide a range of possible values for missing data.

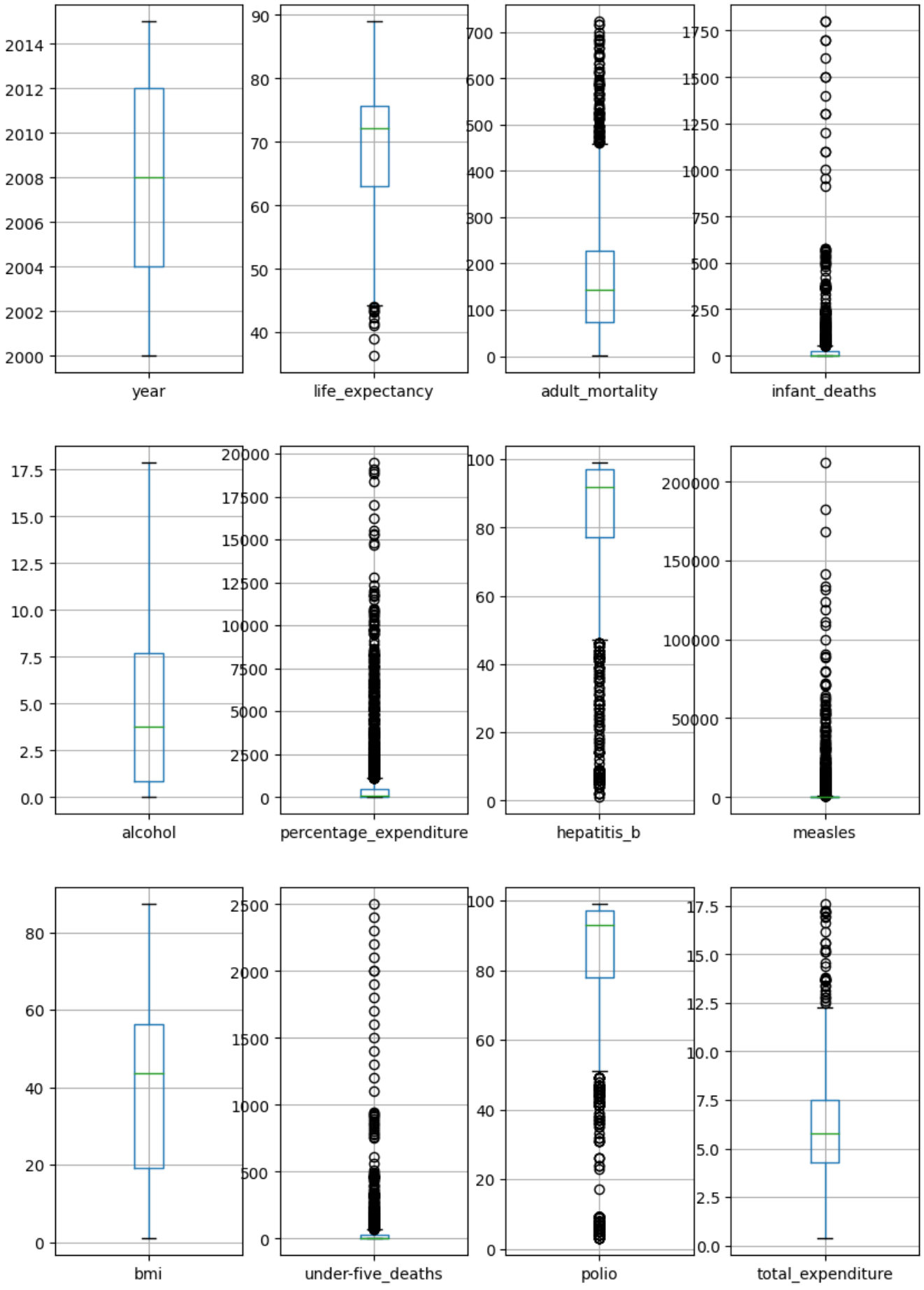
> **REFERENCES**

https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779

`ignoring "palette" because no "hue" variable has been assigned.`

```
#Analysing the numeric variables through boxplot to obtain appropriate imputaion valu

df_impute = df_original.copy()
df_impute.drop(['status','country'], axis=1, inplace=True)
```

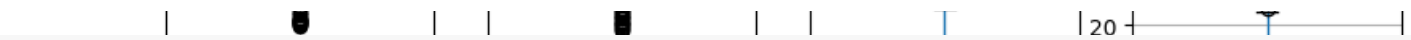
```
plt.figure(figsize=(10,30))

for i, column in enumerate(list(df_impute.columns), start=1):
  plt.subplot(6,4,i)
  df_impute.boxplot(column)
```


year
life_expectancy
adult_mortality
infant_deaths
alcohol
percentage_expenditure
hepatitis_b
measles
bmi
under-five_deaths
polio
total_expenditure

INFERENCE

- In statistics, an **outlier** is a data point that differs significantly from other observations.
- As we can see from the above plots, most of the variables have outliers.
- Imputing the missing values will central average (mean) would therefore not be a good idea.
- Therefore, I would fill the values with the median of the numeric variables/features.
- We will be handling **outliers** later in this notebook

Performing Imputation by filling the null values of the feature with the corresponding median obtained over the years



```
#IMPUTATION
df_original_impt = df_original.copy()
#df_original_impt.info()
#df_original_impt.reset_index(inplace=True)
df_original_impt.groupby('country').apply(lambda group: group.interpolate(method= 'li
df_original_impt
data_imputed=[]
numeric_columns = list(df_impute.columns)
for year in list(df_original_impt.year.unique()):
  #print(f"year ==> {year}")
  year_data = df_original_impt[df_original_impt.year == year].copy()
  #display(year_data)

  for column in numeric_columns:
    #print("inside null col", column)
    year_data[column] = year_data[column].fillna(year_data[column].dropna().median())
  data_imputed.append(year_data)
df_original_impt = pd.concat(data_imputed).copy()

display(df_original_impt)
```

```
Not prepending group keys to the result index of transform-like apply. In the fu
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
```

	country	year	status	life_expectancy	adult_mortality	infant_deaths
0	Afghanistan	2015	Developing	65.0	263.0	62
16	Albania	2015	Developing	77.8	74.0	0
32	Algeria	2015	Developing	75.6	19.0	21
48	Angola	2015	Developing	52.4	335.0	66
64	Antigua and Barbuda	2015	Developing	76.4	13.0	0
...	...	...	...	...	...	...
2873	Venezuela (Bolivarian Republic of)	2000	Developing	72.5	168.0	11
2889	Viet Nam	2000	Developing	73.4	139.0	33
2905	Yemen	2000	Developing	68.0	252.0	48
2921	Zambia	2000	Developing	42.3	614.0	44

OBTAINING THE NULL INFORMATION FROM THE USER-DEFINED FUNCTION: *null_information*

2938 rows × 22 columns

```
#getting null information after imputation
null_information(df_original_impt)
```

```
column : country has 0 null values ---> 0.0 % of nulls
column : year has 0 null values ---> 0.0 % of nulls
column : status has 0 null values ---> 0.0 % of nulls
column : life_expectancy has 0 null values ---> 0.0 % of nulls
column : adult_mortality has 0 null values ---> 0.0 % of nulls
column : infant_deaths has 0 null values ---> 0.0 % of nulls
column : alcohol has 0 null values ---> 0.0 % of nulls
column : percentage_expenditure has 0 null values ---> 0.0 % of nulls
column : hepatitis_b has 0 null values ---> 0.0 % of nulls
column : measles has 0 null values ---> 0.0 % of nulls
column : bmi has 0 null values ---> 0.0 % of nulls
column : under-five_deaths has 0 null values ---> 0.0 % of nulls
```

```
column : polio has 0 null values ---> 0.0 % of nulls
column : total_expenditure has 0 null values ---> 0.0 % of nulls
column : diphtheria has 0 null values ---> 0.0 % of nulls
column : hiv/aids has 0 null values ---> 0.0 % of nulls
column : gdp has 0 null values ---> 0.0 % of nulls
column : population has 0 null values ---> 0.0 % of nulls
column : thinness_1-19_years has 0 null values ---> 0.0 % of nulls
column : thinness_5-9_years has 0 null values ---> 0.0 % of nulls
column : income_composition_of_resources has 0 null values ---> 0.0 % of nulls
column : schooling has 0 null values ---> 0.0 % of nulls
```

```
#plotting heatmap to check the null values after imputation
plt.figure(figsize=(20,10))
sns.heatmap(df_original_impt.isnull(),yticklabels=False,cbar=False,cmap='cividis')
```

```
<Axes: >
```

INFERENCE

> From the above result we see that after performing imputation there are no more null values present in our dataset.

HANDLING OUTLIERS

1. Outliers are extreme values that stand out greatly from the overall pattern of values in a dataset or graph
2. Presence of outliers can hamper our model and statistical analysis from getting the true value as they might distort the result due to their extreme values.
3. Therefore, we handle the outliers for the different features based on the data present in the respective columns

REFERENCES

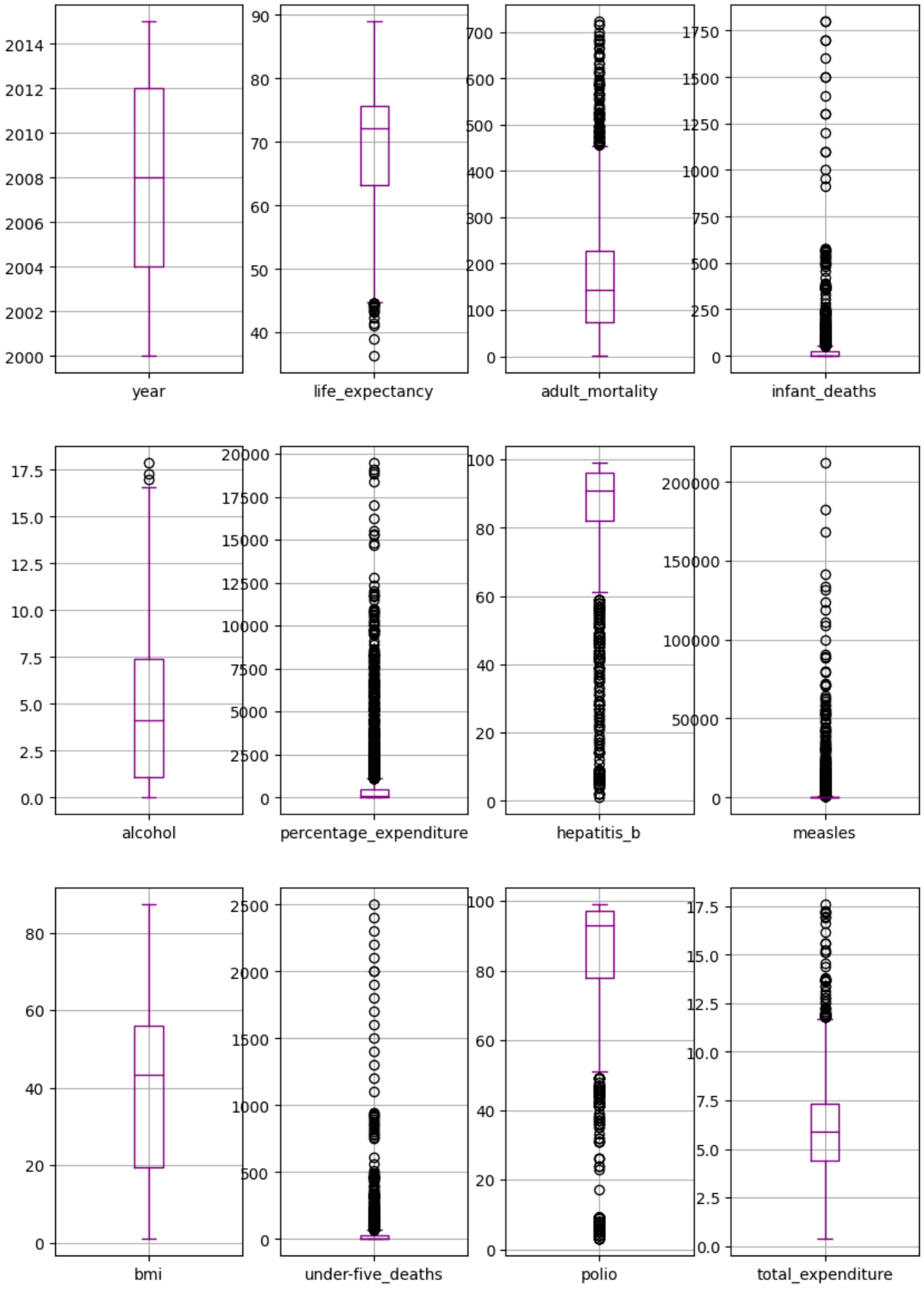
https://statisticsbyjim.com/basics/outliers/

https://www.statisticshowto.com/statistics-basics/find-outliers/

https://www.pluralsight.com/guides/cleaning-up-data-from-outliers

Analysing the variables/features through boxplot to visualize the outliers in our dataset

```
#Analysing the variables/features through boxplot to visualize the outliers in our da

df_outlier_original = df_original_impt.copy()
plt.figure(figsize=(10,30))
numeric_columns = list(df_outlier_original.columns)
numeric_columns.remove("country")
numeric_columns.remove("status")

for i, column in enumerate((numeric_columns), start=1):
  plt.subplot(6,4,i)
  df_outlier_original.boxplot(column, color="purple")
```


year
life_expectancy
adult_mortality
infant_deaths
alcohol
percentage_expenditure
hepatitis_b
measles
bmi
under-five_deaths
polio
total_expenditure

INFERENCE

From the above boxplot we infer the following

- Infant_deaths represents deaths per 1000. Therefore, the data points in the form of outliers which are beyond 1000 don't make sense and are unrealistic. Therefore we will remove those values. Similar analysis is seen for measles, under-five_deaths.
- Many countries spend less than 2500% of their GDP on health. Since GDp is a crucial information which is also used to obtain other info, we will subdtitute the logarithmic value to remove the outlier.
- In real time BMI over 40 is defined as severe obesity. However as per the dataset analysis the median of the bmi for the countries is itself 40. Therefore, we will remove this column and not consider it for our analysis.

```
df_outlier_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   country                          2938 non-null   object 
 1   year                             2938 non-null   int64  
 2   status                           2938 non-null   object 
 3   life_expectancy                  2938 non-null   float64
 4   adult_mortality                  2938 non-null   float64
 5   infant_deaths                    2938 non-null   int64  
 6   alcohol                          2938 non-null   float64
 7   percentage_expenditure           2938 non-null   float64
 8   hepatitis_b                      2938 non-null   float64
 9   measles                          2938 non-null   int64  
 10  bmi                              2938 non-null   float64
 11  under-five_deaths                2938 non-null   int64  
 12  polio                            2938 non-null   float64
 13  total_expenditure                2938 non-null   float64
 14  diphtheria                       2938 non-null   float64
 15  hiv/aids                         2938 non-null   float64
 16  gdp                              2938 non-null   float64
 17  population                       2938 non-null   float64
 18  thinness_1-19_years              2938 non-null   float64
 19  thinness_5-9_years               2938 non-null   float64
 20  income_composition_of_resources  2938 non-null   float64
 21  schooling                        2938 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 527.9+ KB
```

Removing outliers using conditions

```
#removing outliers

#for infant_deaths, measles, under-five_deaths'
df_outlier_original = df_outlier_original[df_outlier_original['infant_deaths']<1001]
df_outlier_original = df_outlier_original[df_outlier_original['measles'] < 1001]
df_outlier_original = df_outlier_original[df_outlier_original['under-five_deaths'] <

#dropping bmi
df_outlier_original.drop(['bmi'],axis=1, inplace=True)
```

Removing outliers using log transformation:

```
#Removing outliers
df_outlier_original['log_percentage_expenditure'] = np.log(df_outlier_original['perce
df_outlier_original['log_population'] = np.log(df_outlier_original['population'])
df_outlier_original['log_gdp'] = np.log(df_outlier_original['gdp'])
df_outlier_original = df_outlier_original.replace([np.inf, -np.inf], 0)

#display(df_outlier_original)
df_outlier_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 24 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   country                          2413 non-null   object 
 1   year                             2413 non-null   int64  
 2   status                           2413 non-null   object 
 3   life_expectancy                  2413 non-null   float64
 4   adult_mortality                  2413 non-null   float64
 5   infant_deaths                    2413 non-null   int64  
 6   alcohol                          2413 non-null   float64
 7   percentage_expenditure           2413 non-null   float64
 8   hepatitis_b                      2413 non-null   float64
 9   measles                          2413 non-null   int64  
 10  under-five_deaths                2413 non-null   int64  
 11  polio                            2413 non-null   float64
 12  total_expenditure                2413 non-null   float64
 13  diphtheria                       2413 non-null   float64
```

```
 14  hiv/aids                         2413 non-null   float64
 15  gdp                              2413 non-null   float64
 16  population                       2413 non-null   float64
 17  thinness_1-19_years              2413 non-null   float64
 18  thinness_5-9_years               2413 non-null   float64
 19  income_composition_of_resources  2413 non-null   float64
 20  schooling                        2413 non-null   float64
 21  log_percentage_expenditure       2413 non-null   float64
 22  log_population                   2413 non-null   float64
 23  log_gdp                          2413 non-null   float64
dtypes: float64(18), int64(4), object(2)
memory usage: 471.3+ KB
divide by zero encountered in log
```

Removing outliers using winsorization

---

1. The winsorization is a method that involves replacing the smallest and largest values of a data set with the observations closest to them.

2. Winsorization is a statistical technique that involves replacing extreme values in a dataset with less extreme values to reduce the impact of outliers on statistical analysis. Here are some key points about Winsorization:

3. The Winsorization process involves identifying extreme values in a dataset and replacing them with less extreme values. For example, the top 5% of values may be replaced with the value at the 95th percentile, and the bottom 5% of values may be replaced with the value at the 5th percentile.

4. Winsorization can be a useful alternative to removing outliers altogether, which can result in loss of information and bias in the analysis.

https://www.statisticshowto.com/winsorize/

```
#removing outliers

df_outlier_original['winz_life_expectancy'] = winsorize(df_outlier_original['life_exp
df_outlier_original['winz_adult_mortality'] = winsorize(df_outlier_original['adult_mo
df_outlier_original['winz_alcohol'] = winsorize(df_outlier_original['alcohol'], (0.0,
df_outlier_original['winz_hepatitis_b'] = winsorize(df_outlier_original['hepatitis_b'
df_outlier_original['winz_polio'] = winsorize(df_outlier_original['polio'], (0.20,0))
df_outlier_original['winz_total_expenditure'] = winsorize(df_outlier_original['total_
df_outlier_original['winz_diphtheria'] = winsorize(df_outlier_original['diphtheria'],
df_outlier_original['winz_income_composition_of_resources'] = winsorize(df_outlier_or
df_outlier_original['winz_schooling'] = winsorize(df_outlier_original['schooling'], (
```

```
df_outlier_original['winz_hiv/aids'] = winsorize(df_outlier_original['hiv/aids'], (0.
df_outlier_original['winz_thinness_1-19_years'] = winsorize(df_outlier_original['thin
df_outlier_original['winz_thinness_5-9_years'] = winsorize(df_outlier_original['thinn

df_outlier_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 36 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   country                               2413 non-null   object 
 1   year                                  2413 non-null   int64  
 2   status                                2413 non-null   object 
 3   life_expectancy                       2413 non-null   float64
 4   adult_mortality                       2413 non-null   float64
 5   infant_deaths                         2413 non-null   int64  
 6   alcohol                               2413 non-null   float64
 7   percentage_expenditure                2413 non-null   float64
 8   hepatitis_b                           2413 non-null   float64
 9   measles                               2413 non-null   int64  
 10  under-five_deaths                     2413 non-null   int64  
 11  polio                                 2413 non-null   float64
 12  total_expenditure                     2413 non-null   float64
 13  diphtheria                            2413 non-null   float64
 14  hiv/aids                              2413 non-null   float64
 15  gdp                                   2413 non-null   float64
 16  population                            2413 non-null   float64
 17  thinness_1-19_years                   2413 non-null   float64
 18  thinness_5-9_years                    2413 non-null   float64
 19  income_composition_of_resources       2413 non-null   float64
 20  schooling                             2413 non-null   float64
 21  log_percentage_expenditure            2413 non-null   float64
 22  log_population                        2413 non-null   float64
 23  log_gdp                               2413 non-null   float64
 24  winz_life_expectancy                  2413 non-null   float64
 25  winz_adult_mortality                  2413 non-null   float64
 26  winz_alcohol                          2413 non-null   float64
 27  winz_hepatitis_b                      2413 non-null   float64
 28  winz_polio                            2413 non-null   float64
 29  winz_total_expenditure                2413 non-null   float64
 30  winz_diphtheria                       2413 non-null   float64
 31  winz_income_composition_of_resources  2413 non-null   float64
 32  winz_schooling                        2413 non-null   float64
 33  winz_hiv/aids                         2413 non-null   float64
 34  winz_thinness_1-19_years              2413 non-null   float64
 35  winz_thinness_5-9_years               2413 non-null   float64
dtypes: float64(30), int64(4), object(2)
memory usage: 697.5+ KB
```

Analysing the variables/features through boxplot after the removal of outliers

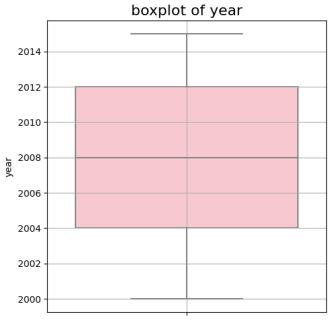
---

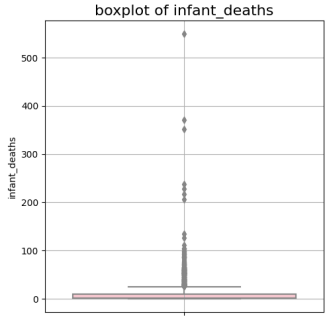
```
columns = list(df_outlier_original.columns)
rem_columns = ['life_expectancy',
 'adult_mortality',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'polio',
 'total_expenditure',
 'diphtheria',
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',
 'schooling','country','status']

required_columns = [i for i in columns if i not in rem_columns ]
print(len(required_columns))
required_columns
```

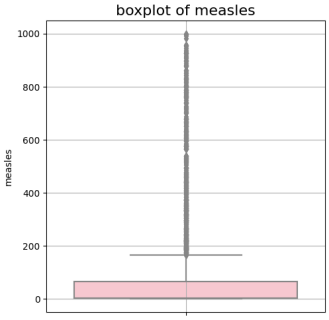
```
19
['year',
 'infant_deaths',
 'measles',
 'under-five_deaths',
 'log_percentage_expenditure',
 'log_population',
 'log_gdp',
 'winz_life_expectancy',
 'winz_adult_mortality',
 'winz_alcohol',
 'winz_hepatitis_b',
 'winz_polio',
 'winz_total_expenditure',
 'winz_diphtheria',
 'winz_income_composition_of_resources',
 'winz_schooling',
 'winz_hiv/aids',
 'winz_thinness_1-19_years',
 'winz_thinness_5-9_years']
```

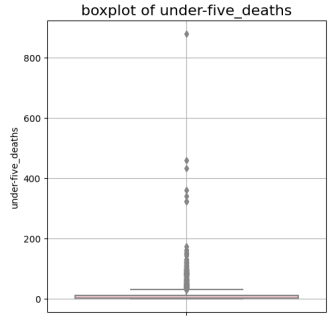
```
plt.figure(figsize=(25,40))
#sns.boxplot(data=data_norm)
i=0

for column in required_columns:
  # i+=1
```

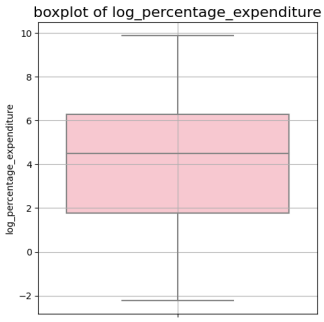
```
  # plt.subplot(7,3,i)
  # #sns.displot(df_combined_new[column], kde=True, bins=60, color='purple')
  # plt.hist(df_outlier_original[column],color="purple")
  # plt.title(f'Histogram of {column}', fontsize=14)
  # plt.grid(True)

  i+=1
  plt.subplot(6,4,i)
  sns.boxplot(y=df_outlier_original[column],color="pink")
  plt.title(f'boxplot of {column}', fontsize=16)
  plt.grid(True)

plt.show()
```

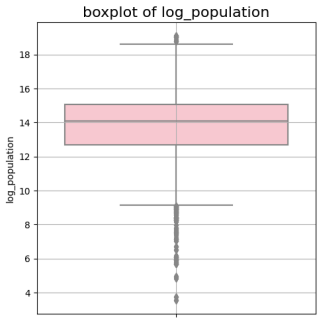

boxplot of year
year

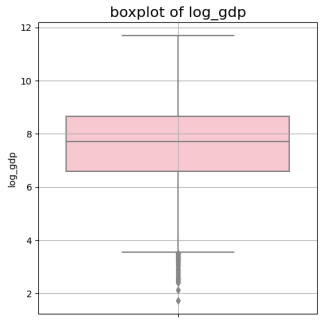

boxplot of infant_deaths
infant_deaths

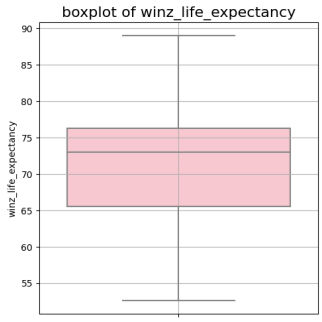

boxplot of measles
measles


boxplot of under-five_deaths
under-five_deaths

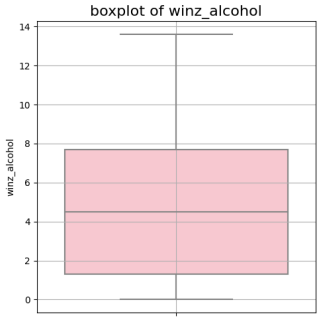

boxplot of log_percentage_expenditure
log_percentage_expenditure

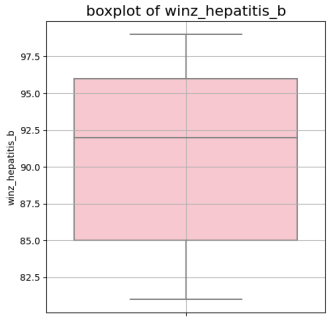

boxplot of log_population
log_population

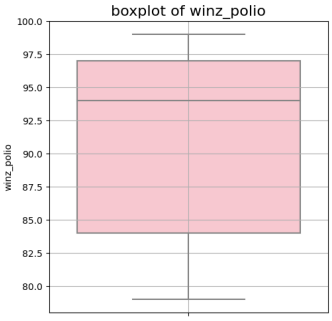

boxplot of log_gdp
log_gdp

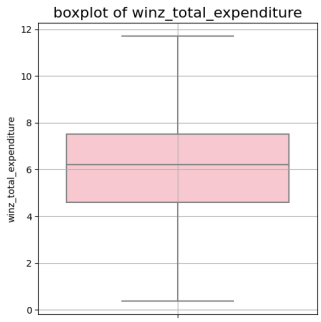

boxplot of winz_life_expectancy
winz_life_expectancy

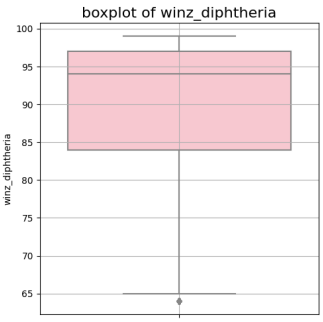

boxplot of winz_adult_mortality
winz_adult_mortality

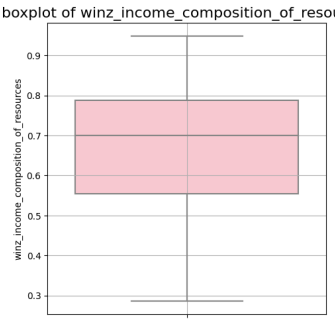

boxplot of winz_alcohol
winz_alcohol

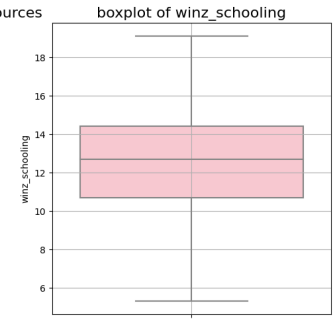

boxplot of winz_hepatitis_b
winz_hepatitis_b

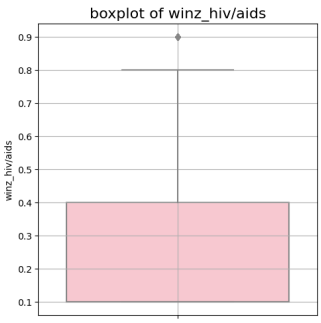

boxplot of winz_polio
winz_polio

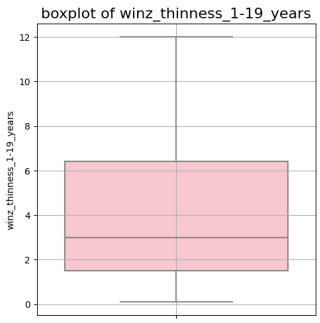

boxplot of winz_total_expenditure
winz_total_expenditure

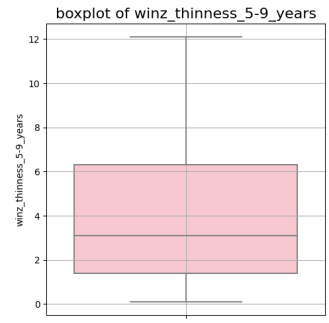

boxplot of winz_diphtheria
winz_diphtheria


boxplot of winz_income_composition_of_resources
winz_income_composition_of_resources


boxplot of winz_schooling
winz_schooling


boxplot of winz_hiv/aids
winz_hiv/aids


boxplot of winz_thinness_1-19_years
winz_thinness_1-19_years


boxplot of winz_thinness_5-9_years
winz_thinness_5-9_years

INFERENCE

From the above boxplot we see that most of the outliers have been removed and our data looks more clean.

PERFORMING ONE-HOT ENCODING ON THE CATEGORICAL DATA COLUMN : STATUS

> One-hot encoding is a technique used in data preprocessing to convert categorical variables into a numerical format that can be used for machine learning models.

> One-hot encoding is useful because many machine learning algorithms cannot handle categorical data directly, and converting them into a numerical format allows for easier analysis and modeling. However, it can also increase the dimensionality of the dataset, which can lead to the "curse of dimensionality" and require more computational resources.

> In our dataset I have performed One-Hot Encoding for the categorical Data Column: `status`

```
df_encoder_original = df_outlier_original

status = pd.get_dummies(df_encoder_original['status'])
#status1 = pd.get_dummies(df_encoder_original['country'])
#status2 = pd.get_dummies(df_encoder_original['year'])
#.drop('Developing',axis=1)
df_encoder_original = pd.concat([df_encoder_original,status], axis=1)
#df_encoder_original = pd.concat([df_encoder_original,status1], axis=1)
#df_encoder_original = pd.concat([df_encoder_original,status2], axis=1)
display(df_encoder_original)

df_encoder_original_model = df_encoder_original
```

	country	year	status	life_expectancy	adult_mortality	infant_deaths
16	Albania	2015	Developing	77.8	74.0	0
32	Algeria	2015	Developing	75.6	19.0	21
48	Angola	2015	Developing	52.4	335.0	66
64	Antigua and Barbuda	2015	Developing	76.4	13.0	0
80	Argentina	2015	Developing	76.3	116.0	8
...	...	...	...	...	...	...
2825	Uruguay	2000	Developing	75.1	131.0	1
2841	Uzbekistan	2000	Developing	67.1	189.0	30
2857	Vanuatu	2000	Developing	69.0	18.0	0
2873	Venezuela (Bolivarian Republic of)	2000	Developing	72.5	168.0	11
2905	Yemen	2000	Developing	68.0	252.0	48

```
df_encoder_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 38 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   country                          2413 non-null   object 
 1   year                             2413 non-null   int64  
 2   status                           2413 non-null   object 
 3   life_expectancy                  2413 non-null   float64
 4   adult_mortality                  2413 non-null   float64
 5   infant_deaths                    2413 non-null   int64  
 6   alcohol                          2413 non-null   float64
 7   percentage_expenditure           2413 non-null   float64
 8   hepatitis_b                      2413 non-null   float64
 9   measles                          2413 non-null   int64  
 10  under-five_deaths                2413 non-null   int64  
 11  polio                            2413 non-null   float64
 12  total_expenditure                2413 non-null   float64
 13  diphtheria                       2413 non-null   float64
 14  hiv/aids                         2413 non-null   float64
 15  gdp                              2413 non-null   float64
 16  population                       2413 non-null   float64
 17  thinness_1-19_years              2413 non-null   float64
```

```
 18  thinness_5-9_years                  2413 non-null   float64
 19  income_composition_of_resources     2413 non-null   float64
 20  schooling                           2413 non-null   float64
 21  log_percentage_expenditure          2413 non-null   float64
 22  log_population                      2413 non-null   float64
 23  log_gdp                             2413 non-null   float64
 24  winz_life_expectancy                2413 non-null   float64
 25  winz_adult_mortality                2413 non-null   float64
 26  winz_alcohol                        2413 non-null   float64
 27  winz_hepatitis_b                    2413 non-null   float64
 28  winz_polio                          2413 non-null   float64
 29  winz_total_expenditure              2413 non-null   float64
 30  winz_diphtheria                     2413 non-null   float64
 31  winz_income_composition_of_resources  2413 non-null   float64
 32  winz_schooling                      2413 non-null   float64
 33  winz_hiv/aids                       2413 non-null   float64
 34  winz_thinness_1-19_years            2413 non-null   float64
 35  winz_thinness_5-9_years             2413 non-null   float64
 36  Developed                           2413 non-null   uint8
 37  Developing                          2413 non-null   uint8
dtypes: float64(30), int64(4), object(2), uint8(2)
memory usage: 702.2+ KB
```

Now we are done with data cleaning. There are no nulls, the outliers have been removed and our dataset is good to go for feature engineering. Now we will go with looking at the data distribution and further normalizing the data.

FEATURE ENGINEERING

> Feature Engineering or data normalization is a method used to normalize the range of independent variables or features of data. So when the values vary a lot in an independent variable, we use feature scaling so that all the values remain in the comparable range.

> https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html

---

DATA DISTRIBUTION

> In Machine Learning, data satisfying Normal Distribution is beneficial for model building. It makes math easier.

Q-Q PLOT

---

1. In a Q-Q plot, the sample quantiles are plotted against the theoretical quantiles of the standard normal distribution on the x-axis and y-axis, respectively. If the sample follows a normal distribution, the Q-Q plot will show the points falling along a straight line. On the other hand, if the sample does not follow a normal distribution, the Q-Q plot will show the points deviating from the straight line.

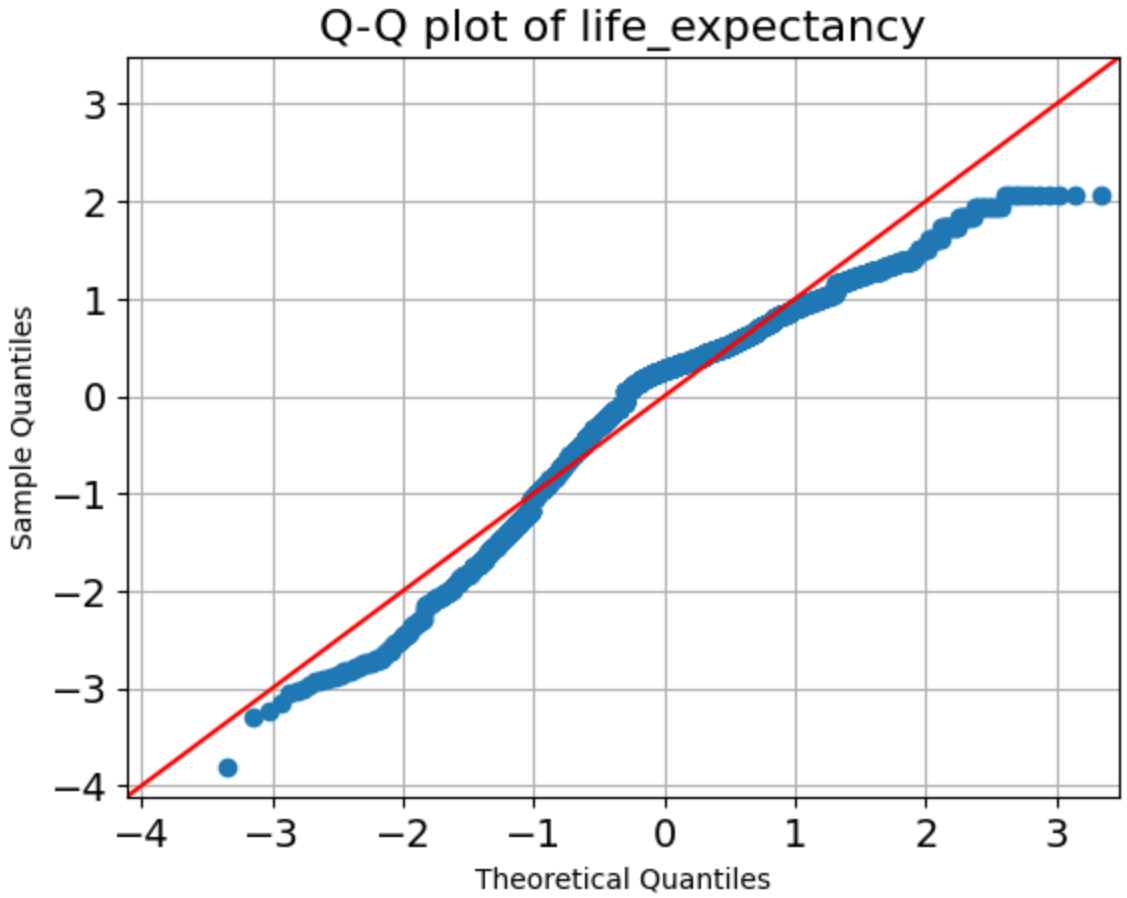
2. Q-Q plots can be used to visually inspect if a dataset follows a normal distribution. This is important because many statistical models, such as linear regression, assume that the data follows a normal distribution. If the data does not follow a normal distribution, it may be necessary to transform the data or use a different statistical model.

3. Q-Q plots can also be used to compare the distribution of two datasets. In this case, the Q-Q plot will show the points for both datasets and if the datasets have similar distributions, the points will fall along a straight line.

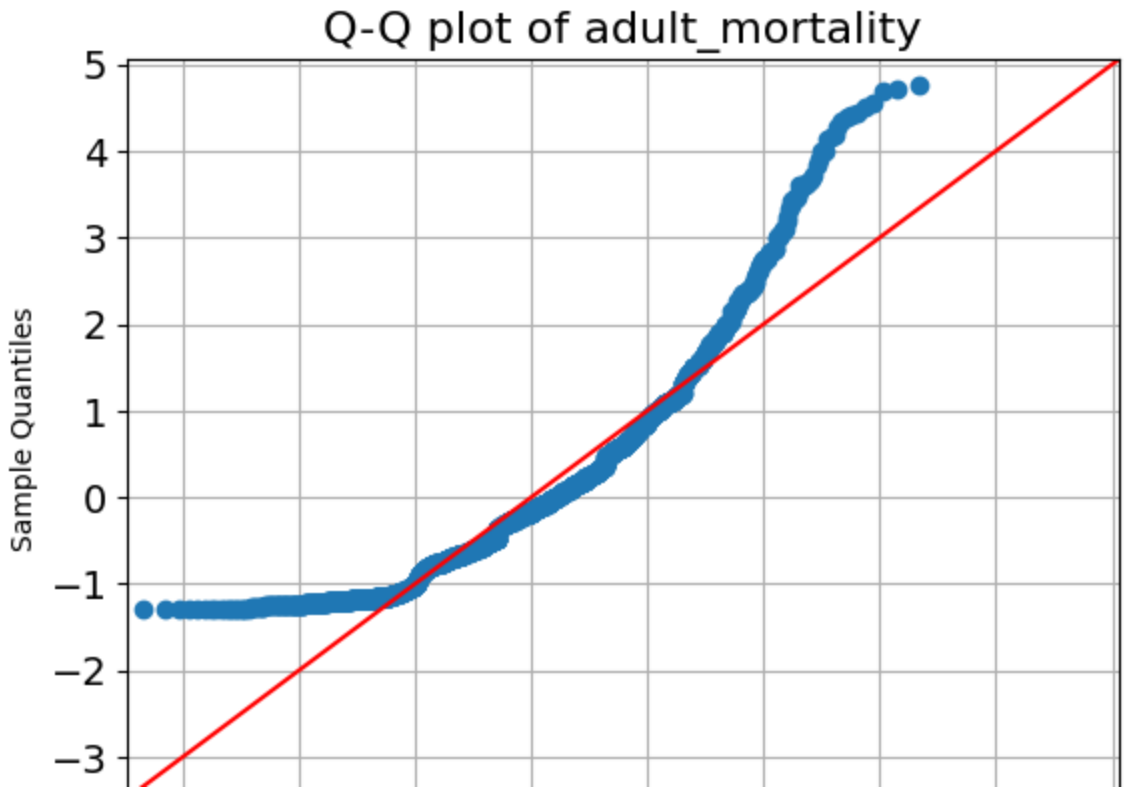
```
#What are the likely distributions of the numeric variables?
#checking the distribution of independent variables
#QQ-PLOT
from statsmodels.graphics.gofplots import qqplot
df_feature_original = df_encoder_original.copy()
df_feature_original.drop(['year','country','status'], axis=1, inplace=True)


#numeric_cols= ['life_expectancy','adult_mortality','alcohol','hepatitis_b','bmi','pc
# data_norm = df_impt[['life_expectancy','adult_mortality','alcohol','hepatitis_b','b
# data_norm = pd.concat([data_norm,df_impt['infant_deaths'],df_impt['hiv/aids'],df_im
# data_norm

for i, columns in enumerate(list(df_feature_original.columns), start=1):
  #print(f"\nFEATURE --> {numeric_cols}")
  #plt.subplot(2,6,i)
  plt.figure(figsize=(10,20))
  print(f"FEATURE --> {columns} and {i}")
  figure=qqplot(df_feature_original[columns],line='45',fit='True')
```

```
  plt.xticks(fontsize=14)
  plt.yticks(fontsize=14)
  plt.title(f"Q-Q plot of {columns}",fontsize=16)
  plt.grid(True)
  plt.show()
```

```
FEATURE --> life_expectancy and 1
<Figure size 1000x2000 with 0 Axes>
```




```
FEATURE --> adult_mortality and 2
<Figure size 1000x2000 with 0 Axes>
```



INFERENCE

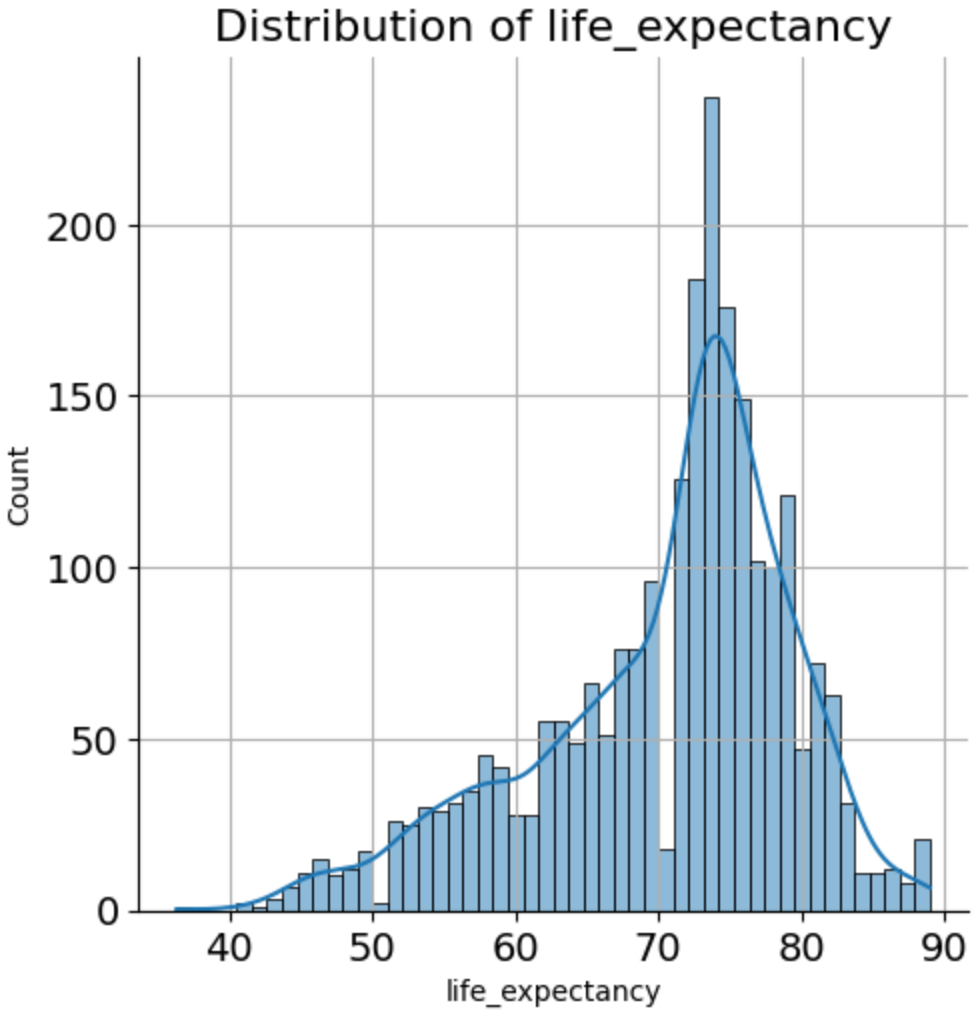
- From the above Q-Q Plot, we can see that almost all the independent variables/features are roughly following normal distribution
- There are few outliers which can be further scaled by using one of the following methods
- StandardScaler
- MinMaxScaler
- RobustScaler
- Normalizer
- We will be using RobustScaler to normalize our dataset furthermore.

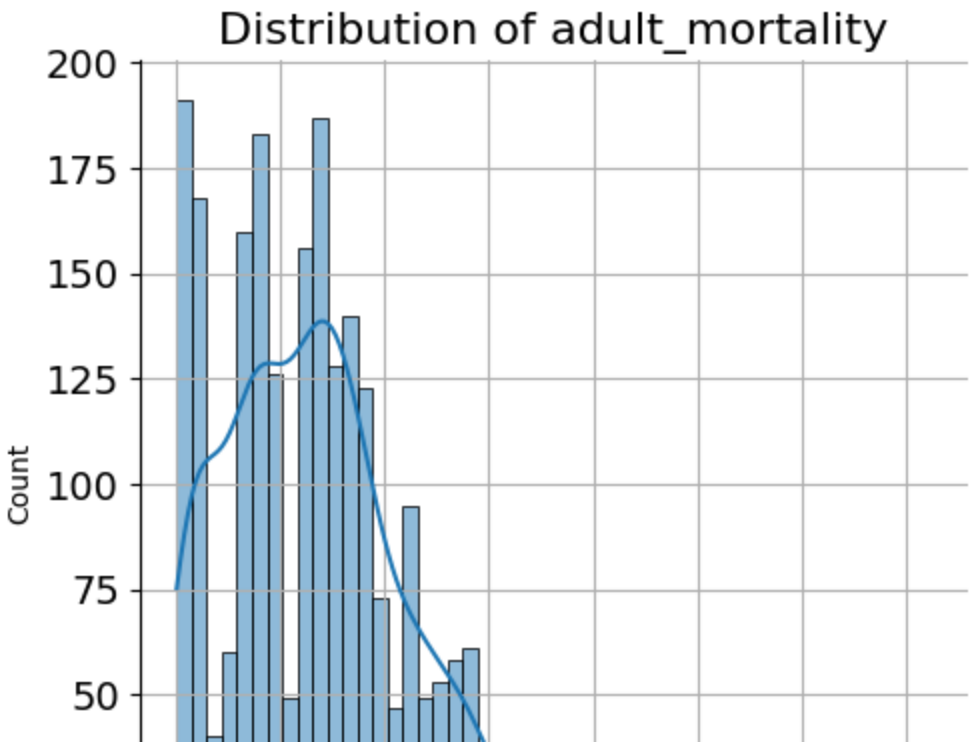
HISTOGRAM PLOT

> A histogram is a chart that plots the distribution of a numeric variable's values as a series of bars. Each bar typically covers a range of numeric values called a bin or class; a bar's height indicates the frequency of data points with a value within the corresponding bin.

```
#histogram viz

#data_norm = df_impt[['life_expectancy','adult_mortality','alcohol','hepatitis_b','bm
plt.figure(figsize=(20,40))
for i,colname in enumerate(list(df_feature_original.columns), start=1):

  sns.displot(df_feature_original[colname], kde=True, bins=50)
  plt.xticks(fontsize=14)
  plt.yticks(fontsize=14)
  plt.title(f"Distribution of {colname}",fontsize=16)
  plt.grid(True)
```

```
More than 20 figures have been opened. Figures created through the pyplot interf
<Figure size 2000x4000 with 0 Axes>
```





INFERENCE

- From the above Histogram Plot, we can see that almost all the independent variables/features are roughly following normal distribution.
- Some are right-skewed like the distribution for polio
- There are few outliers and skweness which can be further scaled by using one of the following methods
- StandardScaler
- MinMaxScaler
- RobustScaler
- Normalizer
- We will be using RobustScaler to normalize our dataset furthermore.

Checking the Ranges of the predictor variables and dependent variable

```
#taking only the clean table columns

old_columns = ['life_expectancy',
 'adult_mortality',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'polio',
 'total_expenditure',
 'diphtheria',
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',
 'schooling']

df_feature_final=df_feature_original.copy()
df_feature_final.drop(old_columns, axis=1, inplace=True)
df_feature_final.info()
```

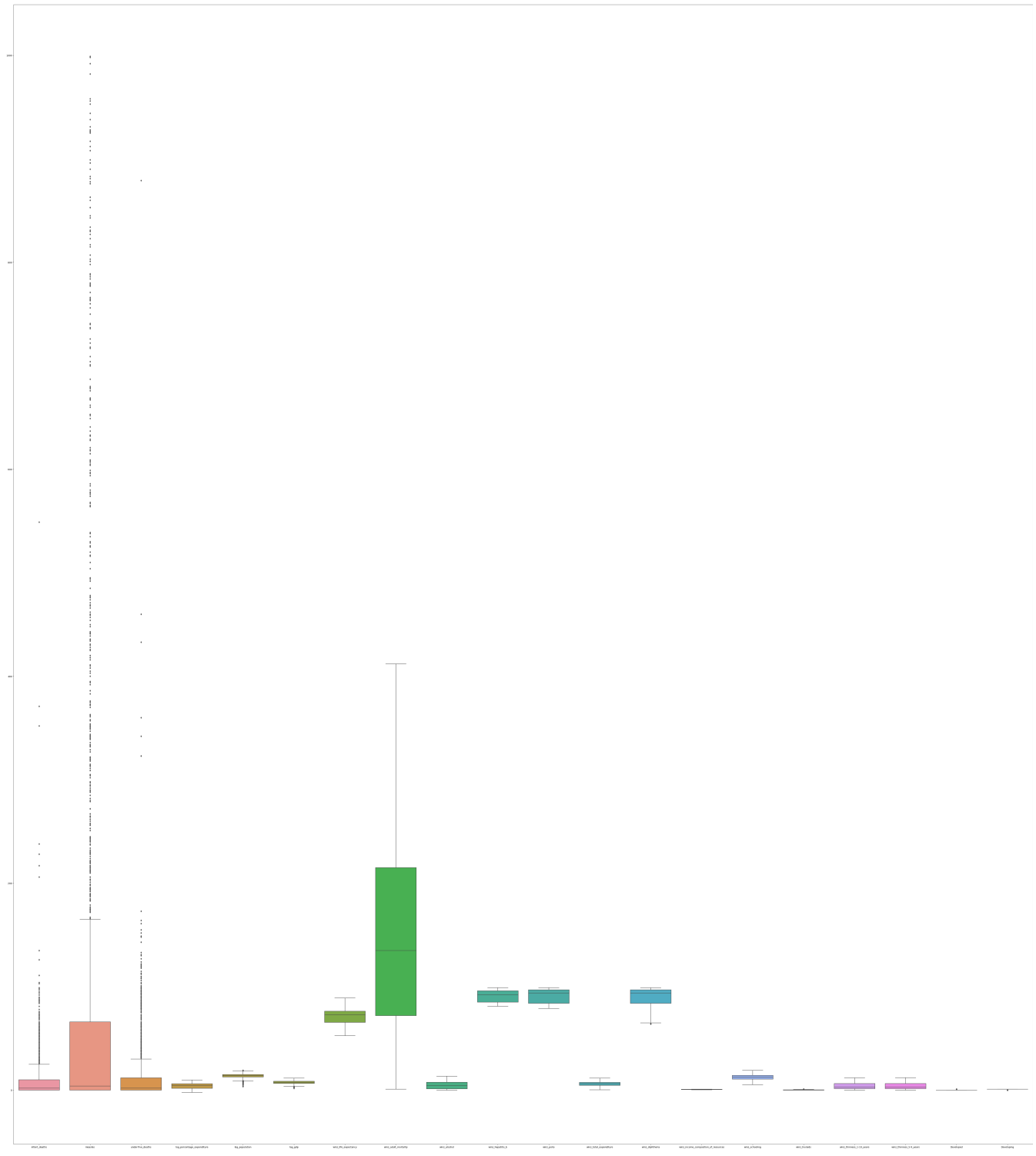
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
```

```
Data columns (total 20 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   infant_deaths                         2413 non-null   int64  
 1   measles                               2413 non-null   int64  
 2   under-five_deaths                     2413 non-null   int64  
 3   log_percentage_expenditure            2413 non-null   float64
 4   log_population                        2413 non-null   float64
 5   log_gdp                               2413 non-null   float64
 6   winz_life_expectancy                  2413 non-null   float64
 7   winz_adult_mortality                  2413 non-null   float64
 8   winz_alcohol                          2413 non-null   float64
 9   winz_hepatitis_b                      2413 non-null   float64
 10  winz_polio                            2413 non-null   float64
 11  winz_total_expenditure                2413 non-null   float64
 12  winz_diphtheria                       2413 non-null   float64
 13  winz_income_composition_of_resources  2413 non-null   float64
 14  winz_schooling                        2413 non-null   float64
 15  winz_hiv/aids                         2413 non-null   float64
 16  winz_thinness_1-19_years              2413 non-null   float64
 17  winz_thinness_5-9_years               2413 non-null   float64
 18  Developed                             2413 non-null   uint8  
 19  Developing                            2413 non-null   uint8  
dtypes: float64(15), int64(3), uint8(2)
memory usage: 362.9 KB
```

Boxplot of the ranges of predictor and dependent variable

```
#checking the ranges of predictor and dependent variables

plt.figure(figsize=(80,90))
sns.boxplot(data=df_feature_final)
```

```
<Axes: >
```



INFERENCE

> Since it is hard to read and visualize the range of distribution of the features, I'll further normalize using one of the normalization technique and then proceed with building the model

PERFORMING NORMALIZATION USING ROBUSTSCALAR

- **RobustScalar** removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

```
df_normalized_original = df_feature_final.copy()
numeric_colums_values = list(df_normalized_original.columns)

df_normalized_final = pd.DataFrame(
    RobustScaler().fit_transform(df_normalized_original[numeric_colums_values]), colu
)
df_normalized_final
df_normalized_final.shape
```

```
(2413, 20)
```

```
df_normalized_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2413 entries, 0 to 2412
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   infant_deaths                   2413 non-null   float64
 1   measles                         2413 non-null   float64
 2   under-five_deaths               2413 non-null   float64
 3   log_percentage_expenditure      2413 non-null   float64
 4   log_population                  2413 non-null   float64
 5   log_gdp                         2413 non-null   float64
 6   winz_life_expectancy            2413 non-null   float64
 7   winz_adult_mortality            2413 non-null   float64
 8   winz_alcohol                    2413 non-null   float64
 9   winz_hepatitis_b                2413 non-null   float64
```

```
 10  winz_polio                          2413 non-null   float64
 11  winz_total_expenditure              2413 non-null   float64
 12  winz_diphtheria                     2413 non-null   float64
 13  winz_income_composition_of_resources  2413 non-null   float64
 14  winz_schooling                      2413 non-null   float64
 15  winz_hiv/aids                       2413 non-null   float64
 16  winz_thinness_1-19_years            2413 non-null   float64
 17  winz_thinness_5-9_years             2413 non-null   float64
 18  Developed                           2413 non-null   float64
 19  Developing                          2413 non-null   float64
dtypes: float64(20)
memory usage: 377.2 KB
```

```
#data after normalizing and feature creation
df_normalized_final.head()
```

	infant_deaths	measles	under-five_deaths	log_percentage_expenditure	log_populati
0	-0.2	-0.060606	-0.166667	0.312007	-1.6116
1	1.9	0.893939	1.833333	-1.004098	1.4324
2	6.4	1.727273	8.000000	-1.004098	0.3121
3	-0.2	-0.060606	-0.166667	-1.004098	0.1882
4	0.6	-0.060606	0.583333	-1.004098	1.4683

IDENTIFYING PREDICTOR SIGNIFICANCE OR FEATURE SELECTION

> **Feature Selection** is the process of selecting the features which are relevant to a machine learning model. It means that you select only those attributes that have a significant effect on the model's output.

> There are many statistical tests which are used to determine the features which are significant to build the model. Some of them are as followed:

- F - Test
- Information gain
- Z-Test

- Correlation test
- ANOVA test
- Chi-square test
- Ordinary Least Square Regression (OLS)

CORRELATION AND P-VALUE

---

1. Correlation is a statistical measure that describes the strength and direction of the linear relationship between two variables.
2. The p-value is a statistical measure that indicates the probability of observing a correlation coefficient as extreme or more extreme than the observed value, assuming that there is no correlation between the two variables.
3. The p-value is used to test the significance of the correlation coefficient by comparing the observed value to the null hypothesis of no correlation.
4. A p-value less than the significance level (typically 0.05) indicates that the observed correlation is statistically significant, and there is evidence against the null hypothesis of no correlation.
5. A higher p-value indicates weaker evidence against the null hypothesis, and a non-significant p-value (greater than 0.05) indicates that there is not enough evidence to reject the null hypothesis of no correlation.
6. A significant p-value does not necessarily imply a strong or meaningful correlation, as other factors such as sample size or outliers can influence the correlation coefficient.
7. Conversely, a non-significant p-value does not necessarily imply no correlation, as there may be other factors that limit the power to detect a true correlation.
8. Therefore, it is important to interpret both the correlation coefficient and the p-value in context and consider other factors that may influence the relationship between the two variables.

REFERENCES

> https://www.analyticsvidhya.com/blog/2021/06/feature-selection-using-statistical-tests/#:~:text=Basically%2C%20the%20p%2Dvalue%20is,to%20reject%20the%20null%20hypothesis.&text=The%20degree%20of%20freedom%20is%20the%20number%20of%20independent%20variables.

> https://sigmamagic.com/blogs/correlation-coefficient/

> https://ranasinghiitkgp.medium.com/implementing-feature-selection-methods-for-machine-learning-bfa2e4b4e02

Distribution of population

CORRELATION TEST

> between two variables. In other words, it helps to determine how closely two variables are related to each other. Herer we will be analysing the same for life expextancy and other variables

> There are several types of correlation tests, but the most commonly used is the Pearson correlation coefficient, which measures the strength of the linear relationship between two continuous variables. The Pearson correlation coefficient ranges from -1 to +1, with -1 indicating a perfect negative correlation (i.e., as one variable increases, the other decreases) and +1 indicating a perfect positive correlation (i.e., as one variable increases, the other also increases).

Correlation between the features after normalization

```
#checking the correlation between all the features in the data
df_normalized_final.corr()
```

	infant_deaths	measles	under-five_deaths	log_perc
infant_deaths	1.000000	0.308503	0.990513	
measles	0.308503	1.000000	0.306626	
under-five_deaths	0.990513	0.306626	1.000000	
log_percentage_expenditure	-0.205785	-0.083086	-0.201167	
log_population	0.234402	0.126869	0.223083	
log_gdp	-0.257234	-0.120302	-0.267517	
winz_life_expectancy	-0.396821	-0.201614	-0.427420	
winz_adult_mortality	0.299522	0.135345	0.321356	
winz_alcohol	-0.189460	-0.142434	-0.188061	
winz_hepatitis_b	-0.192523	-0.135949	-0.204752	
winz_polio	-0.267607	-0.168801	-0.288115	
winz_total_expenditure	-0.066021	-0.069830	-0.069482	
winz_diphtheria	-0.270698	-0.165243	-0.294329	
winz_income_composition_of_resources	-0.308857	-0.163025	-0.332159	
winz_schooling	-0.339566	-0.197212	-0.360625	
winz_hiv/aids	0.306358	0.151786	0.345709	
winz_thinness_1-19_years	0.288396	0.221862	0.299105	

Visualizing Data Distribution after normalization

0.181028 0.039126 0.177397

```
plt.figure(figsize=(25,40))
#sns.boxplot(data=data_norm)
i=0

for column in list(df_normalized_final.columns):
  i+=1
  plt.subplot(10,4,i)
  #sns.displot(df_combined_new[column], kde=True, bins=60, color='purple')
  plt.hist(df_normalized_final[column],color="purple")
  plt.title(f'Histogram of {column}', fontsize=14)
  plt.grid(True)

  i+=1
```

```
  plt.subplot(10,4,i)
  sns.boxplot(y=df_normalized_final[column],color="pink")
  plt.title(f'boxplot of {column}', fontsize=16)
  plt.grid(True)

plt.show()
```


Histogram of infant_deaths
boxplot of infant_deaths
Histogram of measles
boxplot of measles
Histogram of under-five_deaths
boxplot of under-five_deaths
Histogram of log_percentage_expenditure
boxplot of log_percentage_expenditure
Histogram of log_population
boxplot of log_population
Histogram of log_gdp
boxplot of log_gdp
Histogram of winz_life_expectancy
boxplot of winz_life_expectancy
Histogram of winz_adult_mortality
boxplot of winz_adult_mortality
Histogram of winz_alcohol
boxplot of winz_alcohol
Histogram of winz_hepatitis_b
boxplot of winz_hepatitis_b
Histogram of winz_polio
boxplot of winz_polio
Histogram of winz_total_expenditure
boxplot of winz_total_expenditure
Histogram of winz_diphtheria
boxplot of winz_diphtheria
Histogram of winz_income_composition_of_resources
boxplot of winz_income_composition_of_resources
Histogram of winz_schooling
boxplot of winz_schooling
Histogram of winz_hiv/aids
boxplot of winz_hiv/aids
Histogram of winz_thinness_1-19_years
boxplot of winz_thinness_1-19_years
Histogram of winz_thinness_5-9_years
boxplot of winz_thinness_5-9_years
Histogram of Developed
boxplot of Developed
Histogram of Developing
boxplot of Developing

Boxplot of the ranges of predictor and dependent variable after normalizing the data

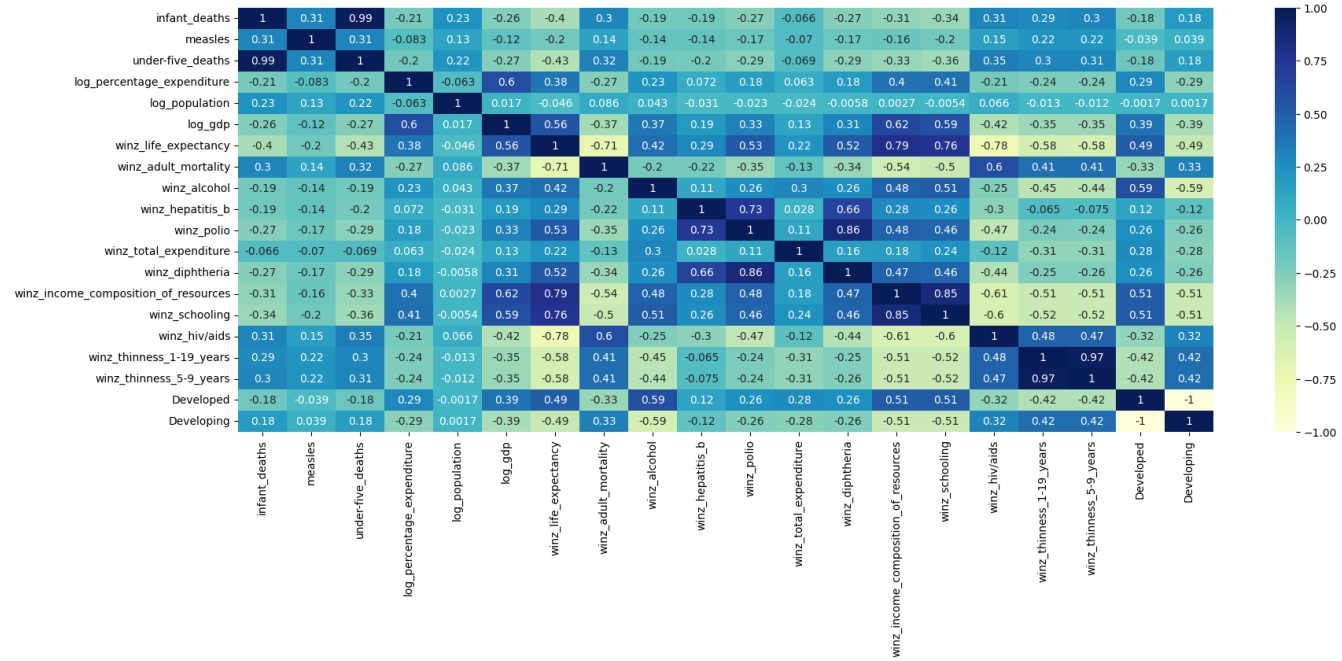

```
#checking the ranges of predictor and dependent variables

plt.figure(figsize=(45,80))
sns.boxplot(data=df_normalized_final)
plt.grid(True)
```


70
60
50
40
30
20
10

Visualizing Heatmap of correlation after the normalization

```
#the heat map of the correlation
plt.figure(figsize=(20,7))
sns.heatmap(df_normalized_final.corr(), annot=True, cmap='YlGnBu')
```

```
<Axes: >
```



Distribution of winz_polio

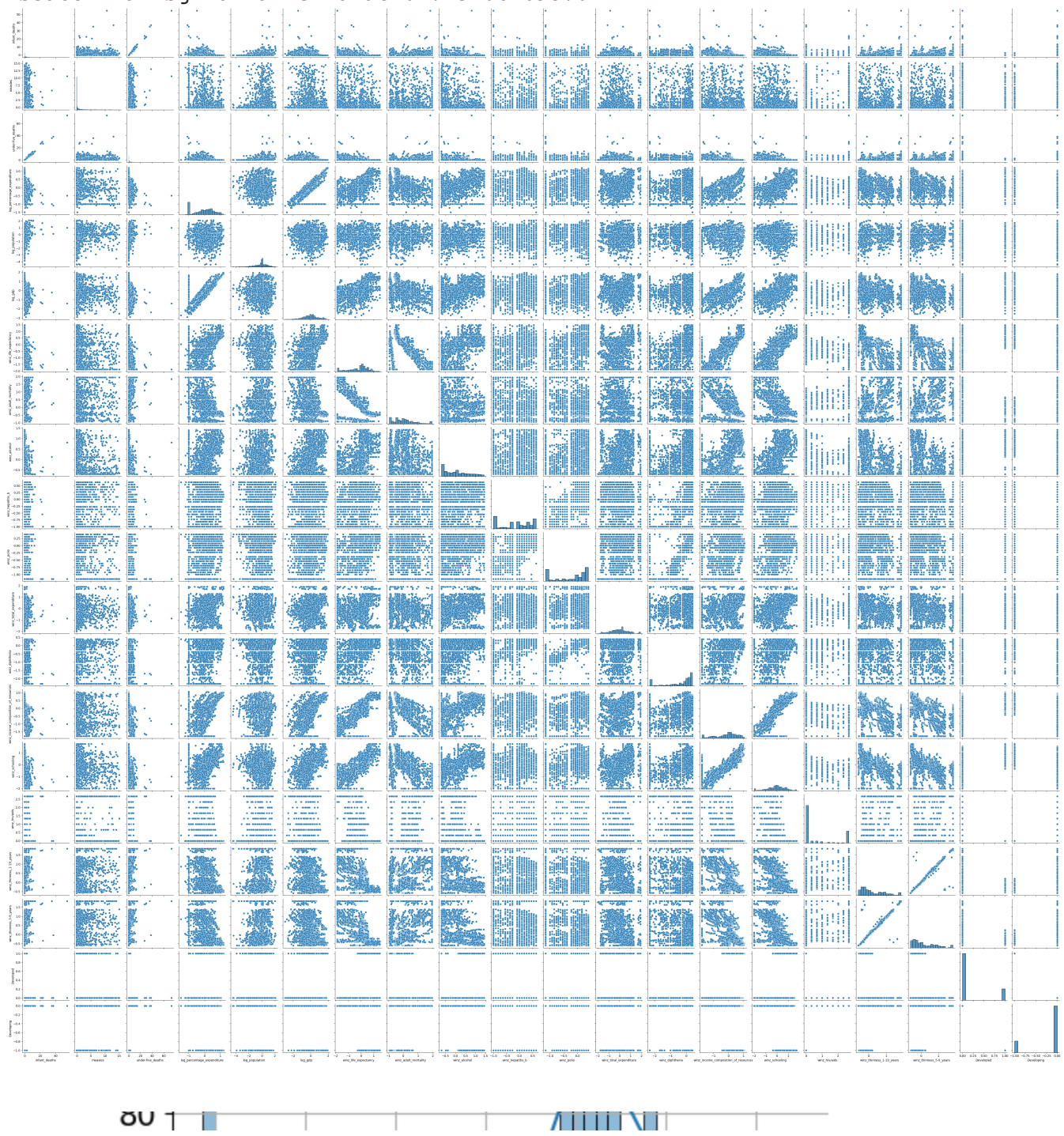
Visualizing Pair plot of the correlation between features after normalization

> A pair plot, also known as a scatterplot matrix, is a visualization technique used to explore the relationship between multiple pairs of continuous variables. In seaborn, a pair plot can be created using the sns.pairplot() function.

> The sns.pairplot() function takes a pandas DataFrame as input and creates a grid of scatterplots and histograms for each pairwise combination of the columns in the DataFrame. The diagonal of the grid shows a histogram of each variable, and the off-diagonal plots show a scatterplot of the two variables.

```
#pair plot to check the colinearity
sns.pairplot(df_normalized_final)
```

```
<seaborn.axisgrid.PairGrid at 0x7f520b169370>
```



INFERENCE

From the above visualizations we can infer the following:

BOXPLOT

- **infant_deaths, measles, under-five_deaths, log_population, log_gdp** have outliers which means that the abnormalities grew higher than expected.

- Features such as **winz_adult_mortality, winz_alcohol, winz_total_expenditure, winz_schooling** appear to be symmetric through the boxplot.
- Features such as **winz_diptheria, winz_polio, winz_thinness** appear to be skewed

> **HEATMAP**

- We see a lot of correlation between the vraiables/features through the heatmap.
- Life expectancy has the highest positive correlation with **income_composition_of_resources** followed by **schooling**, with the coefficient values being **0.79, 0.76**. From this we can infer that as these features increase we see the life_expectancy also to increase.
- Life expectancy has the negative correlation with **adult_mortality** with the coefficient values being **-0.71**. From this we can infer that as adult_mortality decreases we see the life_expectancy to increase.

> **PAIR PLOT**

- In the pair plot, we see that **income_composition_of_resources** and **schooling** have a high positive correlation, which means as the income and availability of a country increases it leads to increase in the schooling of the children.
- In the pair plot, we see that **percentage_expenditure** and **gdp** have a high positive correlation, which means as the gdp of a country increases it leads to increase in the percentage_expenditure of the country.

---

6 8 10 12 14 16 18

ORDINARY LEAST SQUARE REGRESSION TEST (OLS)

---

1. Ordinary Least Squares regression (OLS) is a common technique for estimating coefficients of linear regression equations which describe the relationship between one or more independent quantitative variables and a dependent variable (simple or multiple linear regression). OLS (Ordinary Least Squares) is a statistical method used in machine learning to estimate the parameters of a linear regression model. In simple terms, OLS tries to find the line that best fits a set of data points.

2. The basic idea behind OLS is to minimize the sum of squared differences between the predicted values and the actual values in the data. The line that

minimizes this sum of squared differences is considered the best-fit line and is used to make predictions for new data points.

3. To estimate the parameters of the OLS model, the algorithm calculates the slope and intercept of the best-fit line. The slope represents the change in the dependent variable (y) for every one-unit change in the independent variable (x), while the intercept represents the expected value of y when x equals zero.

4. OLS is widely used in machine learning because of its simplicity and interpretability. It assumes that the relationship between the independent and dependent variables is linear and that the errors in the data are normally distributed with constant variance. If these assumptions are met, OLS can provide accurate and reliable predictions.

REFERENCES

---

https://vitalflux.com/ordinary-least-squares-method-concepts-examples/

https://towardsdatascience.com/understanding-the-ols-method-for-simple-linear-regression-e0a4e8f692cc

---

```
#OLS for finding the p value to check the significant features

df_ols_original = df_normalized_final.copy()
df = df_ols_original.drop(['winz_life_expectancy'], axis=1)
columns = list(df.columns)
#columns
import statsmodels.api as sm

model = sm.OLS(df_ols_original['winz_life_expectancy'],df_ols_original[columns]).fit(

model.summary()
```

OLS Regression Results

Dep. Variable:	winz_life_expectancy	**R-squared (uncentered):**	0.847
Model:	OLS	**Adj. R-squared (uncentered):**	0.846
Method:	Least Squares	**F-statistic:**	734.8
Date:	Mon, 24 Apr 2023	**Prob (F-statistic):**	0.00
Time:	13:24:17	**Log-Likelihood:**	-660.59
No. Observations:	2413	**AIC:**	1357.
Df Residuals:	2395	**BIC:**	1461.
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
infant_deaths	0.0747	0.021	3.642	0.000	0.034	0.115
measles	-0.0004	0.002	-0.184	0.854	-0.005	0.004
under-five_deaths	-0.0763	0.017	-4.444	0.000	-0.110	-0.043
log_percentage_expenditure	0.0296	0.013	2.254	0.024	0.004	0.055
log_population	-0.0028	0.007	-0.401	0.689	-0.017	0.011
log_gdp	0.0387	0.011	3.384	0.001	0.016	0.061
winz_adult_mortality	-0.2444	0.012	-21.143	0.000	-0.267	-0.222
winz_alcohol	0.0003	0.015	0.018	0.986	-0.028	0.029
winz_hepatitis_b	-0.0757	0.017	-4.499	0.000	-0.109	-0.043
winz_polio	0.0739	0.026	2.888	0.004	0.024	0.124
winz_total_expenditure	0.0176	0.009	1.976	0.048	0.000	0.035
winz_diphtheria	0.0559	0.015	3.710	0.000	0.026	0.086
winz_income_composition_of_resources	0.2521	0.019	13.405	0.000	0.215	0.289
winz_schooling	0.0870	0.017	5.214	0.000	0.054	0.120
winz_hiv/aids	-0.1941	0.008	-23.065	0.000	-0.211	-0.178
winz_thinness_1-19_years	0.1196	0.042	2.867	0.004	0.038	0.201
winz_thinness_5-9_years	-0.2060	0.041	-4.999	0.000	-0.287	-0.125
Developed	0.0905	0.010	8.910	0.000	0.071	0.110
Developing	-0.0905	0.010	-8.910	0.000	-0.110	-0.071

Omnibus:	61.094	**Durbin-Watson:**	2.053
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	136.117

INFERENCE

> The **p-value** for each term tests the **null hypothesis** that the **coefficient** is equal to zero (no effect). A **low p-value (< 0.05)** indicates that you can **reject the null hypothesis**. In other words, a predictor that has a low p-value is likely to be a meaningful addition to your model because changes in the predictor's value are related to changes in the response variable.

> Conversely, a larger (insignificant) p-value suggests that changes in the predictor are not associated with changes in the response.

> **RESULT OF OLS TESTING**

```
1. infant_deaths has p-value of 0, which is < 0.05 --> significant feature

2. measles has p-value of 0.8, which is > 0.05 --> insignificant feature

3. under-five_deaths has p-value of 0, which is < 0.05 --> significant feature

4. log_percentage_expenditure has p-value of 0.024, which is < 0.05 --> significant feature

5. log_population has p-value of 0.68, which is > 0.05 --> insignificant feature

6. log_gdp has p-value of 0.001, which is < 0.05 --> significant feature

7. winz_adult_mortality has p-value of 0, which is < 0.05 --> significant feature

8. winz_alcohol has p-value of 0.98, which is > 0.05 --> insignificant feature

9. winz_hepatitis_b has p-value of 0, which is < 0.05 --> significant feature

10. winz_polio has p-value of 0.004, which is < 0.05 --> significant feature

11. winz_total_expenditure has p-value of 0.048, which is < 0.05 --> significant feature

12. winz_diphtheria has p-value of 0, which is < 0.05 --> significant feature

13. winz_income_composition_of_resources  has p-value of 0, which is < 0.05 --> significant

14. winz_schooling has p-value of 0, which is < 0.05 --> significant feature

15. winz_hiv/aids has p-value of 0, which is < 0.05 --> significant feature

16. winz_thinness_1-19_years has p-value of 0.004, which is < 0.05 --> significant feature

17. winz_thinness_5-9_years has p-value of 0, which is < 0.05 --> significant feature
```

```
18. Developed has p-value of 0, which is < 0.05 --> significant feature

19. Developing has p-value of 0, which is < 0.05 --> significant feature
```

F-TEST : f_regression method

---

1. In general, the Regression F-test is used to test whether the regression model fits the data better than the model with no feature.

2. In machine learning, the f_regression method is used to perform univariate feature selection based on the F-test. The F-test is a statistical test that measures the degree of linear dependency between two variables. In feature selection, the F-test is used to evaluate the relationship between each feature and the target variable in a regression problem.

3. The f_regression method takes a set of features and the target variable as input and returns two arrays: scores and pvalues. The scores array contains the F-test scores for each feature, which measure how much the target variable varies with the feature. The higher the F-test score, the more important the feature is in predicting the target variable. The pvalues array contains the corresponding p-values for each F-test score, which measure the significance of the F-test score.

4. The f_regression method can be used to select the most important features in a dataset based on their F-test scores. Features with high F-test scores and low p-values are considered the most important and should be kept in the model, while features with low F-test scores and high p-values can be removed from the model.

```
df_ols_original = df_normalized_final.copy()
df = df_ols_original.drop(['winz_life_expectancy'], axis=1)
columns = list(df.columns)
y = df_ols_original['winz_life_expectancy']
X = df_ols_original[columns]

#from sklearn.feature_selection import SelectKBest, mutual_info_regression
#Select top 2 features based on mutual info regression
selector = SelectKBest(mutual_info_regression, k =10)
```

```
selector.fit(X, y)
X.columns[selector.get_support()]
```

```
Index(['infant_deaths', 'under-five_deaths', 'log_gdp', 'winz_adult_mortality',
       'winz_alcohol', 'winz_income_composition_of_resources',
       'winz_schooling', 'winz_hiv/aids', 'winz_thinness_1-19_years',
       'winz_thinness_5-9_years'],
      dtype='object')
```

```
# from sklearn.datasets import load_boston
# from sklearn.feature_selection import f_regression
# import pandas as pd
# import statsmodels.api as sm
# from pandas.testing import assert_frame_equal

y = df_ols_original['winz_life_expectancy']
X = df_ols_original[columns]

# sklearn f_regression method
F, pval = f_regression(X, y)
F_df = pd.DataFrame({"F_score": F, "p_value": pval})

F_df
```

	F_score	p_value
0	450.609888	8.055823e-92
1	102.155444	1.501190e-23
2	538.914360	9.175623e-108
3	402.457571	6.478224e-83
7	502.436693	3.086049e-101

BUILDING THE MODEL

TRAIN, TEST SPLIT

```
from sklearn.model_selection import train_test_split

insignificang_features = ['measles','log_population','winz_alcohol']
df_model_original = df_normalized_final.copy()

#Removing the insignificant features before modeling
df_model_original.drop(insignificang_features, axis=1, inplace=True)
df_model_original.info()
df = df_model_original.drop(['winz_life_expectancy'], axis=1)
columns = list(df.columns)
X = df_model_original[columns]
y = df_model_original['winz_life_expectancy']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_sta
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2413 entries, 0 to 2412
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   infant_deaths                   2413 non-null   float64
 1   under-five_deaths               2413 non-null   float64
 2   log_percentage_expenditure      2413 non-null   float64
 3   log_gdp                         2413 non-null   float64
 4   winz_life_expectancy            2413 non-null   float64
 5   winz_adult_mortality            2413 non-null   float64
 6   winz_hepatitis_b                2413 non-null   float64
 7   winz_polio                      2413 non-null   float64
 8   winz_total_expenditure          2413 non-null   float64
 9   winz_diphtheria                 2413 non-null   float64
```

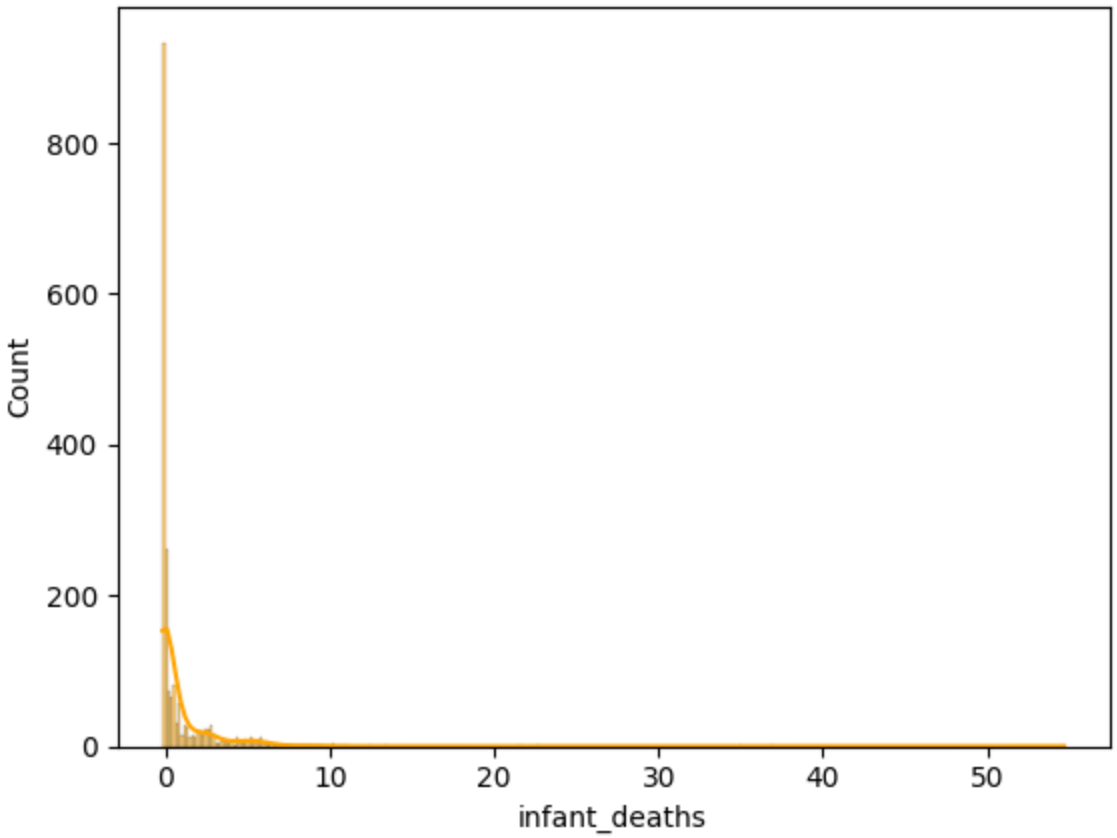
```
 10  winz_income_composition_of_resources  2413 non-null   float64
 11  winz_schooling                        2413 non-null   float64
 12  winz_hiv/aids                         2413 non-null   float64
 13  winz_thinness_1-19_years              2413 non-null   float64
 14  winz_thinness_5-9_years               2413 non-null   float64
 15  Developed                             2413 non-null   float64
 16  Developing                            2413 non-null   float64
dtypes: float64(17)
memory usage: 320.6 KB
```

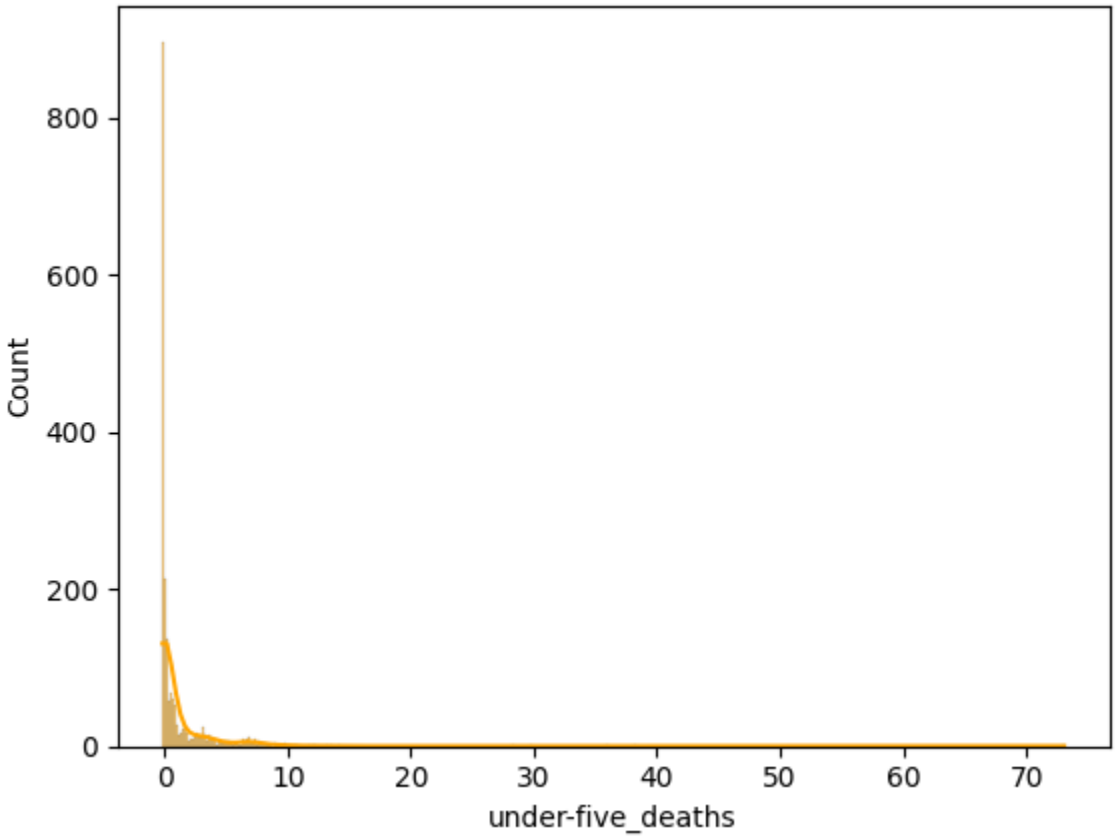
```
X_train.head()
```

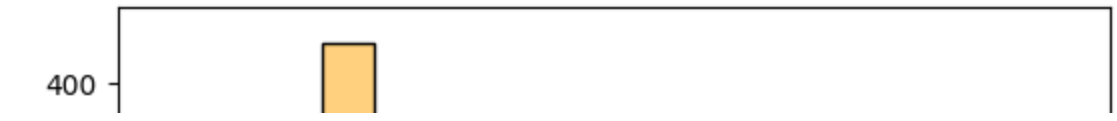
	infant_deaths	under-five_deaths	log_percentage_expenditure	log_gdp	winz_adul
1399	0.2	0.250000	0.077637	-0.144998	
87	0.6	0.833333	-1.004098	-0.316143	
1384	-0.2	-0.166667	-1.004098	-0.146484	
1104	2.8	3.750000	0.104342	-0.324276	
589	4.7	7.333333	-0.247203	-0.845440	

DISTRIBUTION OF TRAINING DATA

```
for i, col in enumerate(X_train.columns):
    plt.figure(i)
    sns.histplot(X_train[col], kde=True, color="orange")
```

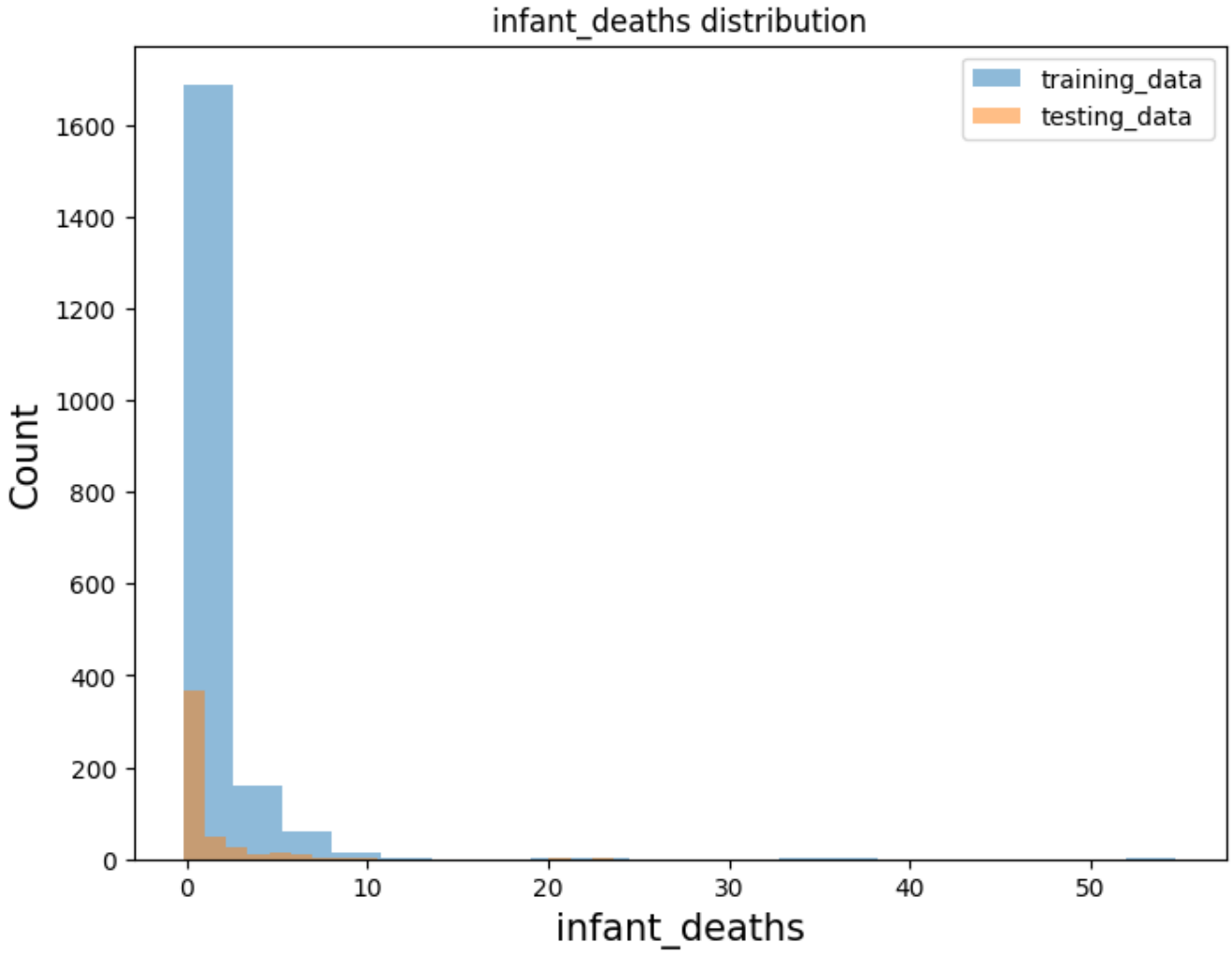

800
600
400
200
0
Count
0
10
20
30
40
50
infant_deaths


800
600
400
200
0
Count
0
10
20
30
40
50
60
70
under-five_deaths


400

COMPARING TRAINING AND TESTING DATA

```
#Training and test data Comparison
for columns in list(X_train.columns):
  plt.figure(figsize=(8,6))
  plt.hist(X_train[columns], bins=20, alpha=0.5, label='training_data')
  plt.hist(X_test[columns], bins=20, alpha=0.5, label='testing_data')
  plt.xlabel(columns, size=15)
  plt.ylabel("Count", size=15)
  plt.title(f'{columns} distribution')
  plt.legend(loc="upper right")
  plt.show()
```


infant_deaths distribution
training_data
testing_data
Count
0
200
400
600
800
1000
1200
1400
1600
0
10
20
30
40
50
infant_deaths

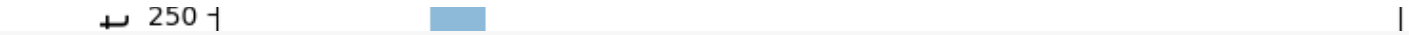

under-five_deaths distribution
training_data
testing_data
Count
250
500
750
1000
1250
1500
1750

INFERENCE

> We see that the training and testing data are evenly spread and contain equivalent range of data throughout

LINEAR REGRESSION

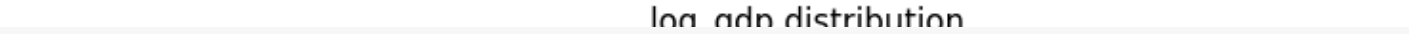
Fit the model using the training data



```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import datasets, linear_model


lm = LinearRegression()
lm.fit(X_train, y_train)
```

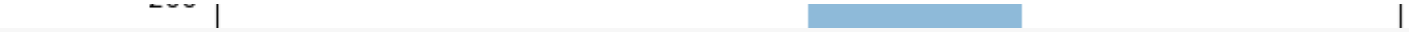
```
LinearRegression()
```

Model Coefficients




```
#PRINT COEFFICIENTS OF THE MODEL
lm.coef_
```

```
array([ 0.0422546 , -0.04959723,  0.04172283,  0.03288417, -0.23656306,
       -0.08466796,  0.07444841,  0.02992643,  0.07197487,  0.26754924,
        0.05708232, -0.22463297,  0.10418632, -0.20825339,  0.05082298,
       -0.05082298])
```



```
#COEFFICIENT OF THE INDEPENDANT VARIABLES
cdf = pd.DataFrame(lm.coef_,X.columns, columns=['Coeff'])
cdf
```

	Coeff
infant_deaths	0.042255
under-five_deaths	-0.049597
log_percentage_expenditure	0.041723
log_gdp	0.032884
winz_adult_mortality	-0.236563
winz_hepatitis_b	-0.084668
winz_polio	0.074448
winz_total_expenditure	0.029926
winz_diphtheria	0.071975
winz_income_composition_of_resources	0.267549
winz_schooling	0.057082
winz_hiv/aids	-0.224633
winz_thinness_1-19_years	0.104186
winz_thinness_5-9_years	-0.208253
Developed	0.050823
Developing	-0.050823

PREDICTIONS AND EVALUATION

TRAINING DATASET

```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

y_predictions = lm.predict(X_train)

from sklearn import metrics

print('Mean Absolute Error [MAE]: ', round(metrics.mean_absolute_error(y_train,y_pred
print('Mean Square Error [MSE]: ', round(metrics.mean_squared_error(y_train, y_predic
print('Root mean Square Error [RMSE]: ', round(np.sqrt(metrics.mean_squared_error(y_t
print('\nCoefficient of Determination :', round(r2_score(y_train,y_predictions)))
```

```
r2 = r2_score(y_train,y_predictions)
print('R^2 score on training set =',r2)
```

```
Mean Absolute Error [MAE]:  0.23
Mean Square Error [MSE]:  0.1
Root mean Square Error [RMSE]:  0.313

Coefficient of Determination : 1
R^2 score on training set = 0.8394553636541762
```

TEST DATASET

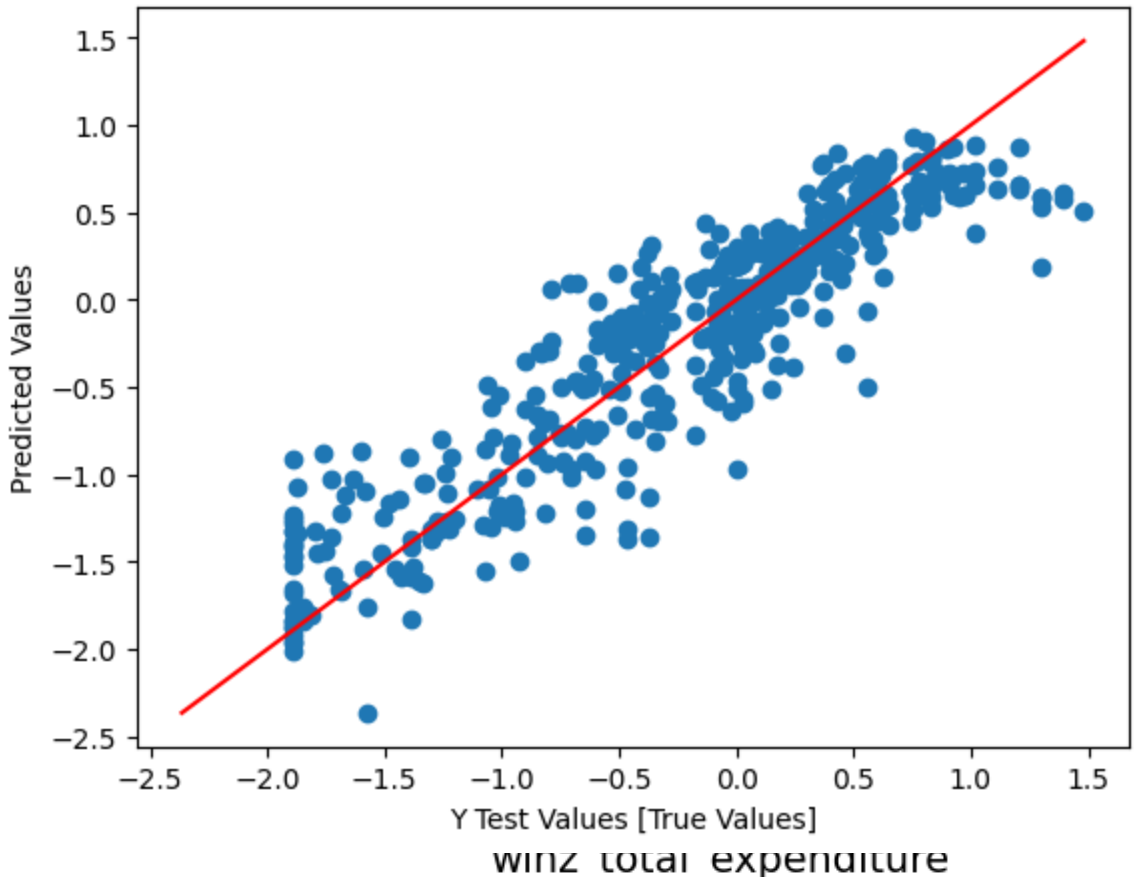
```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

y_predictions = lm.predict(X_test)

from sklearn import metrics

print('Mean Absolute Error [MAE]: ', round(metrics.mean_absolute_error(y_test,y_predi
print('Mean Square Error [MSE]: ', round(metrics.mean_squared_error(y_test, y_predict
print('Root mean Square Error [RMSE]: ', round(np.sqrt(metrics.mean_squared_error(y_t
print('\nCoefficient of Determination :', round(r2_score(y_test,y_predictions)))

r2 = r2_score(y_test,y_predictions)
print('R^2 score on training set =',r2)
```

```
Mean Absolute Error [MAE]:  0.25
Mean Square Error [MSE]:  0.11
Root mean Square Error [RMSE]:  0.326

Coefficient of Determination : 1
R^2 score on training set = 0.8331176308978094
```

SCATTER PLOT OF THE ORIGINAL TEST VALUES VERSUS THE PREDICTED VALUES

```
#SCATTER PLOT OF THE REAL TEST VALUES VERSUS THE PREDICTED VALUES
plt.scatter(y_test, y_predictions)
plt.xlabel('Y Test Values [True Values]')
plt.ylabel('Predicted Values')

p1 = max(max(y_predictions), max(y_test))
p2 = min(min(y_predictions), min(y_test))
plt.plot([p1, p2], [p1, p2], 'r-')
```

```
[<matplotlib.lines.Line2D at 0x7f522ae19730>]
```



wiiz total expenditure

We see that the model build is fairly alog the best fitted line and is seen to be significantly colinear

EXPLAINED VARIANCE or R-SQUARE

> **EXPLAINED VARIANCE:** The explained variation is the sum of the squared of the differences between each predicted y-value and the mean of y.

$$\text{Explained Variance} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

> **R-squared** R-squared is a measure of how well the independent variable(s) can predict the dependent variable. It ranges from 0 to 1, where a value of 1 indicates that all of the variance in the dependent variable is explained by the independent

> variable(s). It's often used as a goodness-of-fit measure for regression models.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

> It is a very important concept to understand how much information we can lose by reconciling the dataset.

REFERENCES

> https://corporatefinanceinstitute.com/resources/data-science/r-squared/

```
metrics.explained_variance_score(y_test, y_predictions)
```

```
0.8331181521198573
```

RESIDUALS

> A residual is the difference between an observed value and a predicted value in regression analysis.

> Residual = Observed value – Predicted value

$$\hat{e_i} = y_i - \hat{y_i}$$

> The Histogram of the Residual can be used to check whether the variance is normally distributed. A symmetric bell-shaped histogram which is evenly distributed around zero indicates that the normality assumption is likely to be true.

REFERENCES

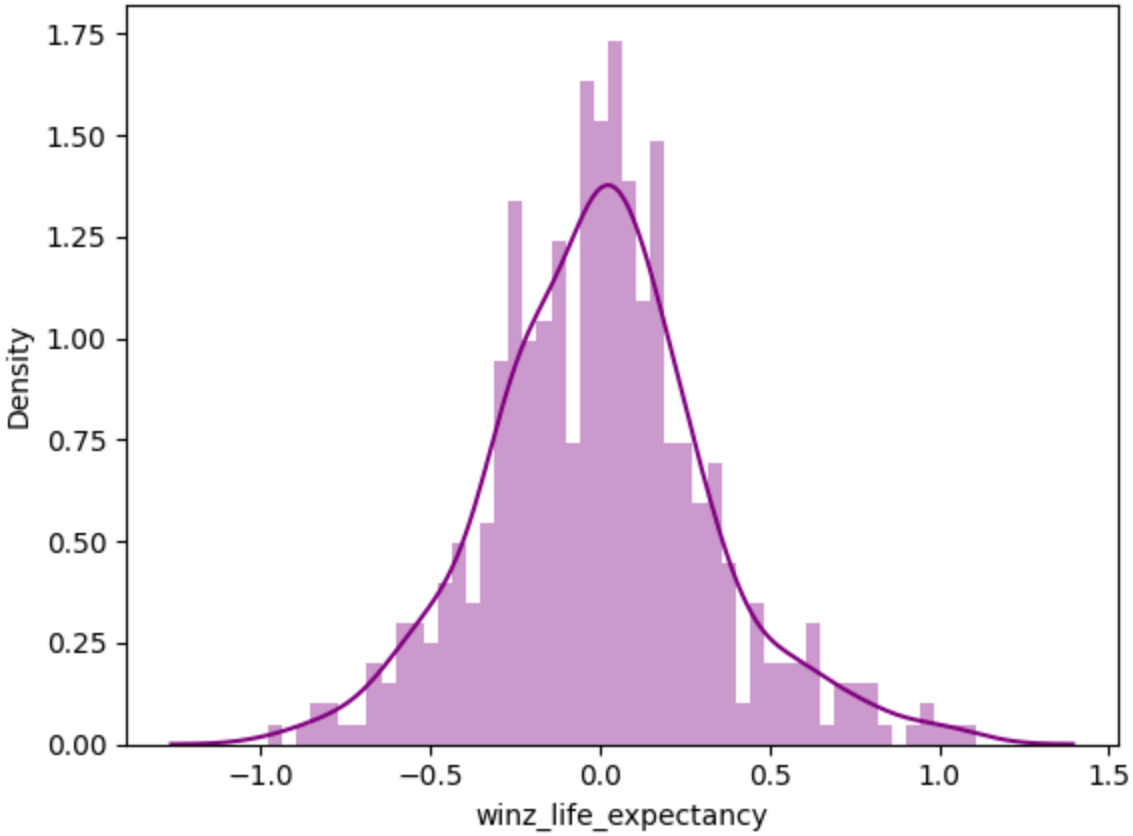
> https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/residuals.html#:~:text=Definition,yi%E2%88%92%5Eyi.

> https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/regression-library/a/introduction-to-residuals

---

```
#RESIDUALS

sns.distplot((y_test-y_predictions), bins=50, color='purple')
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

<Axes: xlabel='winz_life_expectancy', ylabel='Density'>
```



UNDERSTANDING THE IMPORTANT FEATURES

```
#Understanding the important features
import eli5
from eli5.sklearn import PermutationImportance
perm = PermutationImportance(lm, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Weight	Feature
0.1722 ± 0.0247	winz_hiv/aids
0.1447 ± 0.0272	winz_income_composition_of_resources
0.0971 ± 0.0327	winz_adult_mortality
0.0629 ± 0.0084	under-five_deaths
0.0587 ± 0.0066	winz_thinness_5-9_years
0.0208 ± 0.0030	infant_deaths
0.0184 ± 0.0053	winz_thinness_1-19_years
0.0139 ± 0.0025	winz_diphtheria
0.0091 ± 0.0021	winz_schooling
0.0074 ± 0.0037	winz_hepatitis_b
0.0047 ± 0.0052	winz_polio
0.0032 ± 0.0009	Developed
0.0031 ± 0.0017	log_percentage_expenditure
0.0023 ± 0.0008	Developing
0.0022 ± 0.0032	log_gdp
0.0009 ± 0.0025	winz_total_expenditure

PERMUTATION MODEL

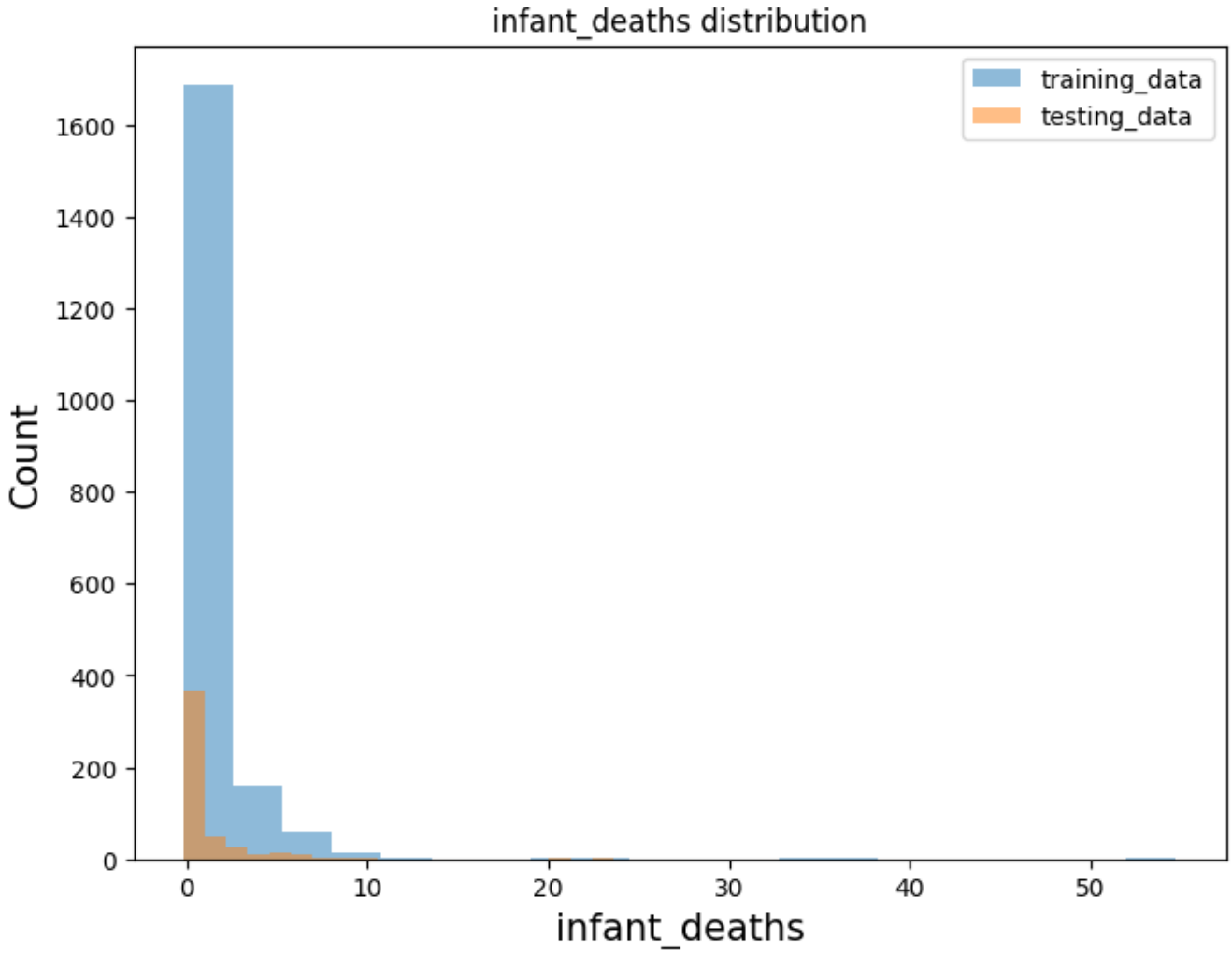
> Permutation importance is calculated after a model has been fitted.

> The values towards the top are the most important features, and those towards the bottom matter least.

> The values towards the top are the most important features, and those towards the bottom matter least.

Analysis of data in training and test sets

```
#Training and test data Comparison
for columns in list(X_train.columns):
  plt.figure(figsize=(8,6))
  plt.hist(X_train[columns], bins=20, alpha=0.5, label='training_data')
  plt.hist(X_test[columns], bins=20, alpha=0.5, label='testing_data')
  plt.xlabel(columns, size=15)
  plt.ylabel("Count", size=15)
  plt.title(f'{columns} distribution')
  plt.legend(loc="upper right")
  plt.show()
```


infant_deaths distribution
training_data
testing_data
Count
0
200
400
600
800
1000
1200
1400
1600
0
10
20
30
40
50
infant_deaths


under-five_deaths distribution
training_data
testing_data
Count
250
500
750
1000
1250
1500
1750

Ranges of the predictor variables

> From the above visualizations we can infer the following:

> **BOXPLOT**

- **infant_deaths, measles, under-five_deaths, log_population, log_gdp** have outliers which means that the abnormalities grew higher than expected.
- Features such as **winz_adult_mortality, winz_alcohol, winz_total_expenditure, winz_schooling** appear to be symmetric through the boxplot.
- Features such as **winz_diptheria, winz_polio, winz_thinness** appear to be skewed

> **HEATMAP**

- We see a lot of correlation between the vraiables/features through the heatmap.
- Life expectancy has the highest positive correlation with **income_composition_of_resources** followed by **schooling**, with the coefficient values being **0.79, 0.76**. From this we can infer that as these features increase we see the life_expectancy also to increase.
- Life expectancy has the negative correlation with **adult_mortality** with the coefficient values being **-0.71**. From this we can infer that as adult_mortality decreases we see the life_expectancy to increase.

> **PAIR PLOT**

- In the pair plot, we see that **income_composition_of_resources** and **schooling** have a high positive correlation, which means as the income and availability of a country increases it leads to increase in the schooling of the children.
- In the pair plot, we see that **percentage_expenditure** and **gdp** have a high positive correlation, which means as the gdp of a country increases it leads to increase in the percentage_expenditure of the country.

Distributions of the predictor variables

> **Q-Q PLOT**

- From the above Q-Q Plot, we can see that almost all the independent variables/features are roughly following normal distribution
- There are few outliers which can be further scaled by using one of the following methods
- StandardScaler
- MinMaxScaler
- RobustScaler
- Normalizer
- We will be using RobustScaler to normalize our dataset furthermore.

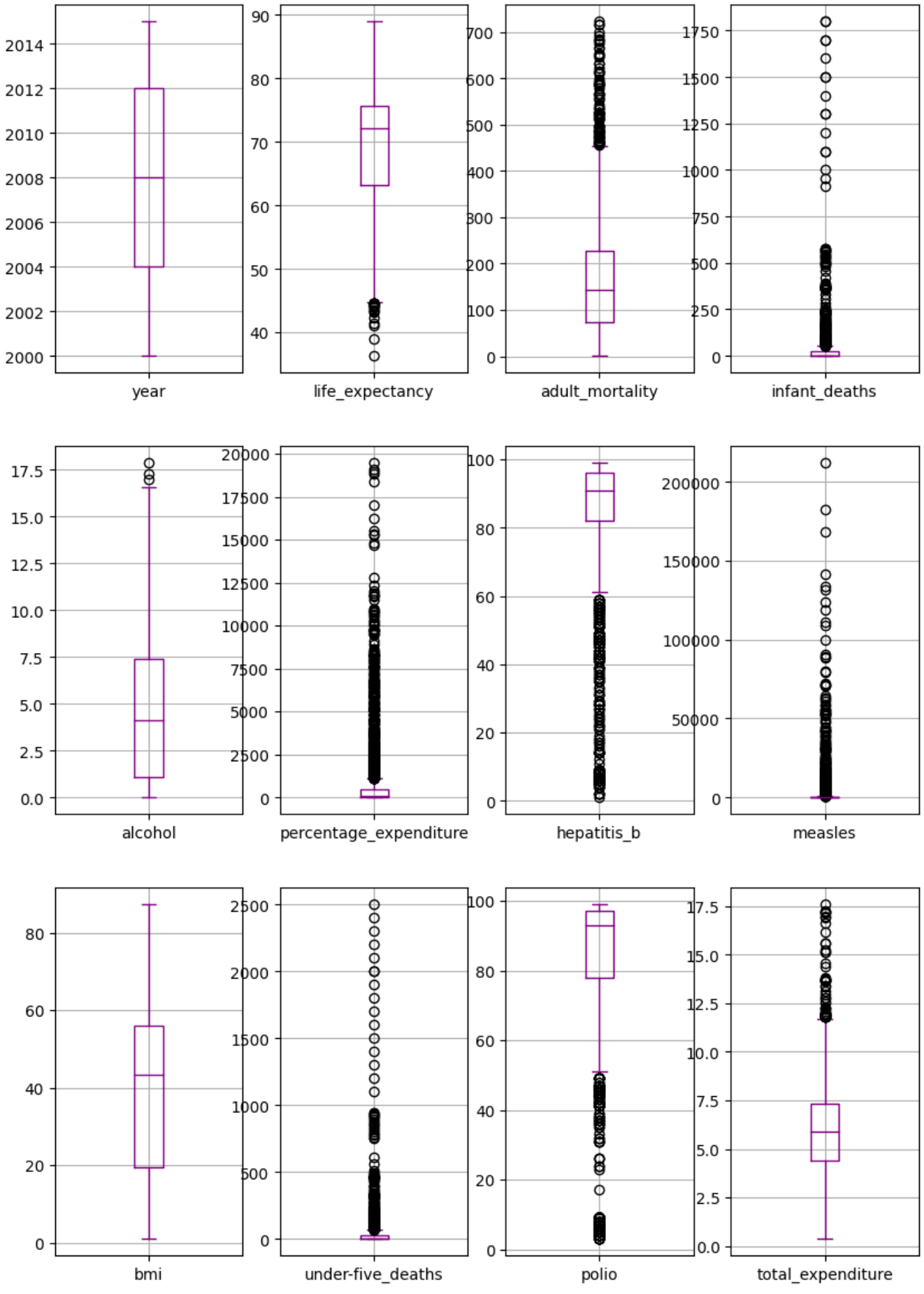
HISTOGRAM HISTOGRAM

- From the above Histogram Plot, we can see that almost all the independent variables/features are roughly following normal distribution.
- Some are right-skewed like the distribution for polio

Analysis of model with and without Outliers

DATASET WITH OUTLIERS

```
#DATASET WITH OUTLIERS
df_original_impt

#Analysing the variables/features through boxplot to visualize the outliers in our da

df_outlier = df_original_impt.copy()
plt.figure(figsize=(10,30))
numeric_columns = list(df_outlier.columns)
numeric_columns.remove("country")
numeric_columns.remove("status")

for i, column in enumerate((numeric_columns), start=1):
  plt.subplot(6,4,i)
  df_outlier.boxplot(column, color="purple")
```


year
life_expectancy
adult_mortality
infant_deaths
alcohol
percentage_expenditure
hepatitis_b
measles
bmi
under-five_deaths
polio
total_expenditure

```
df_outlier.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   country                          2938 non-null   object 
 1   year                             2938 non-null   int64  
 2   status                           2938 non-null   object 
 3   life_expectancy                  2938 non-null   float64
 4   adult_mortality                  2938 non-null   float64
 5   infant_deaths                    2938 non-null   int64  
 6   alcohol                          2938 non-null   float64
 7   percentage_expenditure           2938 non-null   float64
 8   hepatitis_b                      2938 non-null   float64
 9   measles                          2938 non-null   int64  
 10  bmi                              2938 non-null   float64
 11  under-five_deaths                2938 non-null   int64  
 12  polio                            2938 non-null   float64
 13  total_expenditure                2938 non-null   float64
 14  diphtheria                       2938 non-null   float64
 15  hiv/aids                         2938 non-null   float64
 16  gdp                              2938 non-null   float64
 17  population                       2938 non-null   float64
 18  thinness_1-19_years              2938 non-null   float64
 19  thinness_5-9_years               2938 non-null   float64
 20  income_composition_of_resources  2938 non-null   float64
 21  schooling                        2938 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 527.9+ KB
```

ONE-HOT ENCODING

```
from sklearn.model_selection import train_test_split

#Removing the insignificant features before modeling
insignificang_features = ['measles','population','alcohol','country','year','bmi']

df_outlier_train = df_outlier.copy()
df_outlier_train.drop(insignificang_features, axis=1, inplace=True)



# One-hot encoding a single column
from sklearn.preprocessing import OneHotEncoder
```

```
ohe = OneHotEncoder()
trans = ohe.fit_transform(df_outlier_train[['status']])

# Getting one hot encoded categories
print(ohe.categories_)

df_outlier_train[ohe.categories_[0]] = trans.toarray()
df_outlier_train.head()
df_outlier_train.drop(['status'], axis=1, inplace=True)
df_outlier_train.info()
df_outlier_train.head()
```

```
[array(['Developed', 'Developing'], dtype=object)]
<class 'pandas.core.frame.DataFrame'>
```

MODEL BUILDING WITH OUTLIERS

```
0   life expectancy                  2938 non-null   float64
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import datasets, linear_model


model_outlier = LinearRegression()
model_no_outlier = LinearRegression()
```

```
df = df_outlier_train.drop(['life_expectancy'], axis=1)
columns = list(df.columns)
X = df_outlier_train[columns]
y = df_outlier_train['life_expectancy']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_sta

model_outlier.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
X_train.head()
```

	adult_mortality	infant_deaths	percentage_expenditure	hepatitis_b	u five_d
1865	147.0	3	321.613259	98.0	
2812	117.0	0	482.803945	94.0	
1789	22.0	61	4.632776	75.0	
1779	43.0	93	47.172507	89.0	
765	325.0	2	94.133029	86.0	

MODEL BUILDING WITHOUT OUTLIERS

```
from sklearn.model_selection import train_test_split

insignificang_features = ['measles','log_population','winz_alcohol']
df_model_original = df_normalized_final.copy()

#Removing the insignificant features before modeling
df_model_original.drop(insignificang_features, axis=1, inplace=True)
#df_model_original.info()
df = df_model_original.drop(['winz_life_expectancy'], axis=1)
columns = list(df.columns)
X1 = df_model_original[columns]
y1 = df_model_original['winz_life_expectancy']


X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size = 0.3, rand
model_no_outlier.fit(X_train1, y_train1)
```

▾ LinearRegression

LinearRegression()

TEST AND EVALUATION

WITH OUTLIERS

```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

y_predictions = model_outlier.predict(X_test)

from sklearn import metrics

print('Mean Absolute Error [MAE]: ', round(metrics.mean_absolute_error(y_test,y_predi
print('Mean Square Error [MSE]: ', round(metrics.mean_squared_error(y_test, y_predict
print('Root mean Square Error [RMSE]: ', round(np.sqrt(metrics.mean_squared_error(y_t
print('\nCoefficient of Determination :', round(r2_score(y_test,y_predictions)))

r2 = r2_score(y_test,y_predictions)
print('R^2 score on testing set =',r2)
```

```
Mean Absolute Error [MAE]:  3.02
Mean Square Error [MSE]:  16.5
Root mean Square Error [RMSE]:  4.062

Coefficient of Determination : 1
R^2 score on testing set = 0.8061939160365815
```

WITHOUT OUTLIERS

```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

y_predictions1 = model_no_outlier.predict(X_test1)

from sklearn import metrics

print('Mean Absolute Error [MAE]: ', round(metrics.mean_absolute_error(y_test1,y_pred
print('Mean Square Error [MSE]: ', round(metrics.mean_squared_error(y_test1, y_predic
print('Root mean Square Error [RMSE]: ', round(np.sqrt(metrics.mean_squared_error(y_t
print('\nCoefficient of Determination :', round(r2_score(y_test1,y_predictions1)))

r2 = r2_score(y_test1,y_predictions1)
print('R^2 score on testing set =',r2)
```

```
Mean Absolute Error [MAE]:  0.25
Mean Square Error [MSE]:  0.11
Root mean Square Error [RMSE]:  0.325

Coefficient of Determination : 1
R^2 score on testing set = 0.8372565558534919
```

INFERENCE

> We see a signifant decrease in the root mean square of the model predicted without the outliers.

> With outliers --> RMSE --> **4.06**

> Without outliers --> RMSE --> **0.325**

CALCULATING MSE AND VARIANCE USING USER-DEFINED FUNCTION AND SCIKIT BUILT-IN FUNCTION

EXPECTED ERROR

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

COMPONENTS OF PREDICTIVE ERRORS

> **Bias:** Difference between the prediction of the true model and the average models (models build on n number of samples obtained from the population).

> **Variance:** Difference between the prediction of all the models obtained from the sample with the average model.

> **Irreducible Error** It is the irreducible error that a model cannot predict.

REFERENCES

- https://www.analyticsvidhya.com/blog/2020/12/a-measure-of-bias-and-variance-an-experiment/
- https://towardsdatascience.com/simple-mathematical-derivation-of-bias-variance-error-4ab223f28791
- https://www.bmc.com/blogs/bias-variance-machine-learning/
- https://medium.com/analytics-vidhya/calculation-of-bias-variance-in-python-8f96463c8942

MEAN SQUARE ERROR, VARIANCE OF RESIDUALS

> ### RESIDUAL

> A residual is the difference between an observed value and a predicted value in a regression model.

> **Residual** = Observed value – Predicted value

$$e_i = y_i - \hat{y_i}$$

MEAN SQUARE ERROR

Mean squared error is calculated by squaring the residual errors of each data point, summing the squared errors, and dividing the sum by the total number of data points.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

REFERENCES

https://study.com/learn/lesson/mean-squared-error-formula.html

---

RESIDUAL VARIANCE

In a **regression model**, the residual variance is defined as the sum of squared differences between predicted data points and observed data points.

It is calculated as:

$$Var(e_i) = \frac{1}{n}\sum_{i=1}^{n}(e_i - \bar{e})^2$$

REFERENCES

https://www.statology.org/how-to-interpret-residual-standard-error/

```
#NORMALIZED DATAFRAME
df_model_impute=df_normalized_final.copy()
```

```
#FEATURE SELECTION
feature = 'winz_life_expectancy'
```

Generating Null values using the **fill_na** functions as defined above

---

```
df_fill_na = fill_na(df_model_impute, feature)
```

```
winz_life_expectancy
```

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	
...	...	...	
2408	0.194444	0.194444	
2409	-0.546296	-0.546296	
2410	-0.370370	-0.370370	
2411	-0.046296	-0.046296	
2412	-0.462963	-0.462963	

2413 rows × 4 columns

```
PERCENTAGE OF NULL VALUES BEFORE ADDING Nan
```

	column_name	percent_missing
winz_life_expectancy	winz_life_expectancy	0.0
winz_life_expectancy_percent_1	winz_life_expectancy_percent_1	0.0
winz_life_expectancy_percent_5	winz_life_expectancy_percent_5	0.0
winz_life_expectancy_percent_10	winz_life_expectancy_percent_10	0.0

```
PERCENTAGE OF NULL VALUES AFTER ADDING Nan

The pandas.np module is deprecated and will be removed from pandas in a future v
The pandas.np module is deprecated and will be removed from pandas in a future v
The pandas.np module is deprecated and will be removed from pandas in a future v
```

	column_name	percent_missing
winz_life_expectancy	winz_life_expectancy	0.00
winz_life_expectancy_percent_1	winz_life_expectancy_percent_1	0.99
winz_life_expectancy_percent_5	winz_life_expectancy_percent_5	5.01

OBSERVATION

- Using the **fill_na** function which has been defined above, we have created 3 columns **winz_life_expectancy_percent_1, winz_life_expectancy_percent_5, winz_life_expectancy_percent_10** consisting of 1%, 5% and 10% Nan values respectively
- We can see two tables

> - table 1 : consisting of no null values
> - table 2 : consisting of **1%, 5%, 10%** null values

df after adding **1%, 5%, 10%** Nan values in **winz_life_expectancy_percent_1, winz_life_expectancy_percent_5, winz_life_expectancy_percent_10** columns respectively

```
#df after adding Nan values
df_fill_na
```

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	
...	...	...	
2408	0.194444	0.194444	
2409	-0.546296	-0.546296	
2410	-0.370370	-0.370370	
2411	-0.046296	-0.046296	
2412	-0.462963	-0.462963	

2413 rows × 4 columns

IMPUTATION METHOD - MEAN

Filling the Nan values with the **MEAN** imputation

```
df_impute_mean = impute_values(df_fill_na, 'mean')
df_impute_mean.head()
```

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy_pe
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	

```
df_mean_stats = statistical_measures(df_impute_mean)
```

```
MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2
```

MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:

```
Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2
```

```
================================================================================
1% IMPUTATION
================================================================================

MEAN SQUARE ERROR - MSE
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.005392615630677118
MSE USING SCIKIT FUNCTION -->  0.005392615630677118

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  -0.00104506876349503

VARIANCE USING MATHEMATICAL FORMULA -->  0.00539
VARIANCE USING SCIKIT FUNCTION -->  0.00539

================================================================================
5% IMPUTATION
================================================================================
```

```
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.03737677926162646
MSE USING SCIKIT FUNCTION -->  0.03737677926162646

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  -0.007374091394556821

VARIANCE USING MATHEMATICAL FORMULA -->  0.03732
VARIANCE USING SCIKIT FUNCTION -->  0.03734

================================================================================
10% IMPUTATION
================================================================================

MEAN SQUARE ERROR - MSE
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.0702921626871262
MSE USING SCIKIT FUNCTION -->  0.0702921626871262

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  0.004320329034183316

VARIANCE USING MATHEMATICAL FORMULA -->  0.0703
VARIANCE USING SCIKIT FUNCTION -->  0.0703
```

INFERENCE

> We see that the MSE 1% of MEAN imputation is the least, the value being --> **0.006**

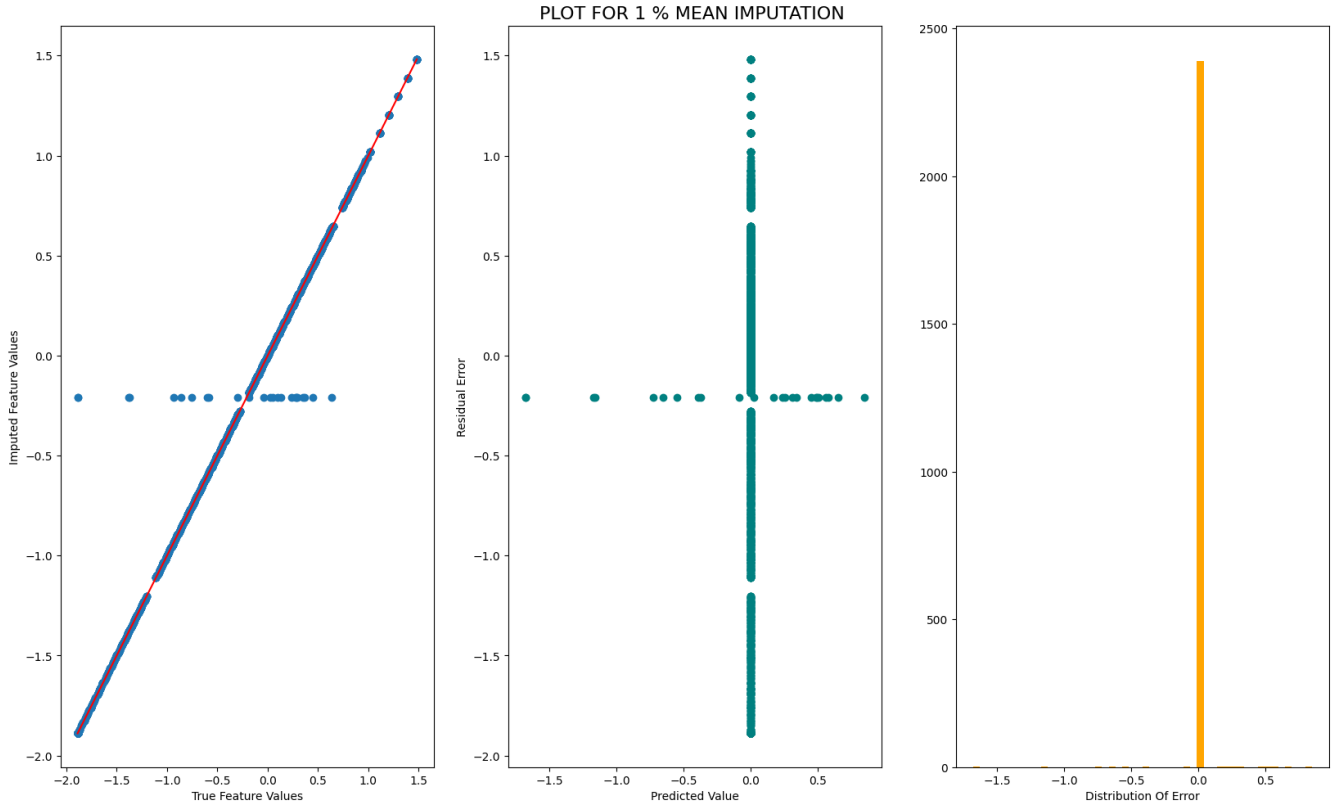
```
df_mean_stats
```

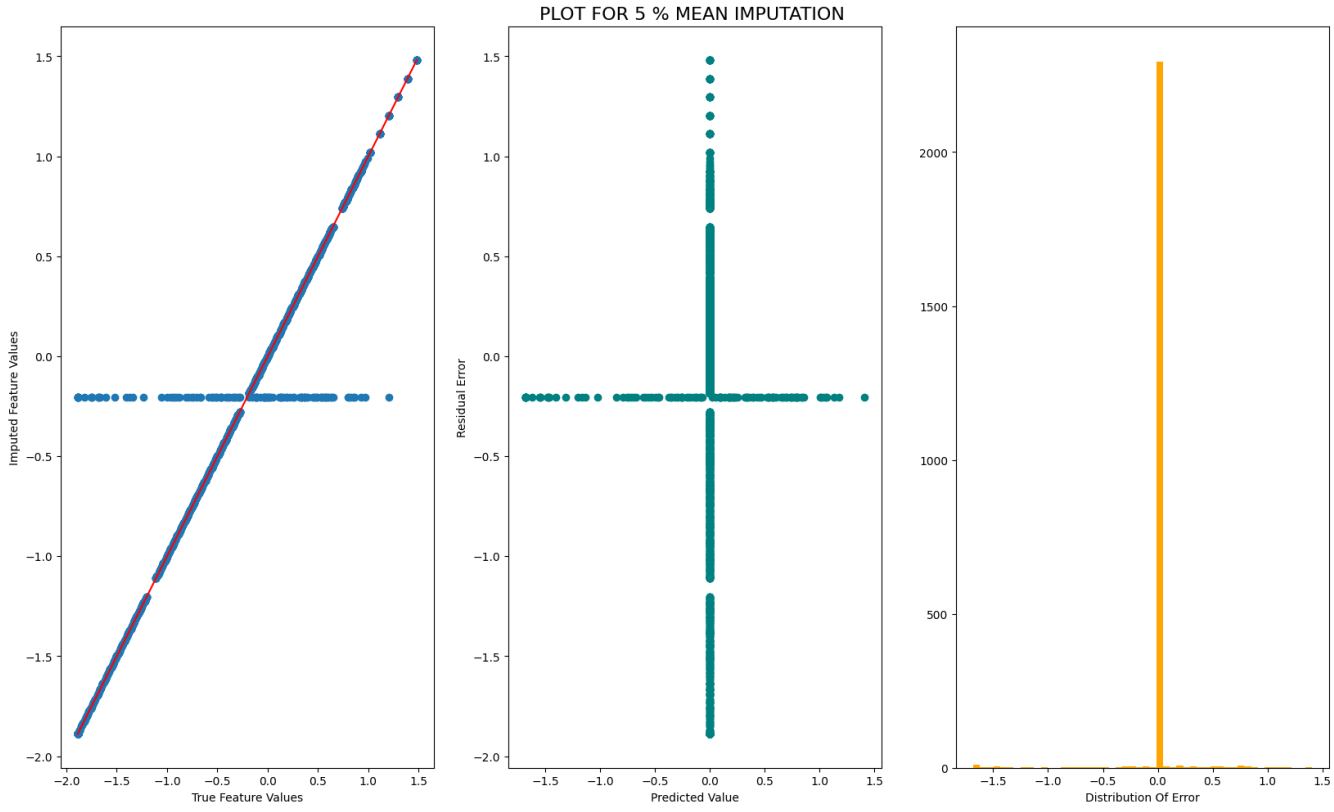
	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	

RESIDUAL PLOT FOR 1%, 5% AND 10% MEAN IMPUTATION

```
residual_plot(df_mean_stats)
```

```
RESIDUAL PLOT FOR 1% IMPUTATION
RESIDUAL PLOT FOR 5% IMPUTATION
RESIDUAL PLOT FOR 10% IMPUTATION
```





INFERENCE

> We see that the Residual plots of all these graphs tell us that the difference between the actual and the imputed value is negligible and therefore we can choose this method for imputation for the current dataset.

IMPUTATION METHOD - MEDIAN

Filling the Nan values with the **MEDIAN** imputation

```
df_impute_median = impute_values(df_fill_na, 'median')
df_impute_median.head()
```

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy_pe
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	

```
df_median_stats = statistical_measures(df_impute_median)
```

```
MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2
```

MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:

```
Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2
```

==========

```
MEAN SQUARE ERROR - MSE
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.006273389396825724
MSE USING SCIKIT FUNCTION -->  0.006273389396825724

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  -0.0031388620282113873

VARIANCE USING MATHEMATICAL FORMULA -->  0.00626
VARIANCE USING SCIKIT FUNCTION -->  0.00627

================================================================================
5% IMPUTATION
================================================================================

MEAN SQUARE ERROR - MSE
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.04247874027117632
MSE USING SCIKIT FUNCTION -->  0.04247874027117632

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  -0.017612929962702032

VARIANCE USING MATHEMATICAL FORMULA -->  0.04217
VARIANCE USING SCIKIT FUNCTION -->  0.04219

================================================================================
10% IMPUTATION
================================================================================

MEAN SQUARE ERROR - MSE
--------------------------------------------------------------------------------
MSE USING MATHEMATICAL FORMULA -->  0.07308140149635296
MSE USING SCIKIT FUNCTION -->  0.07308140149635296

VARIANCE
--------------------------------------------------------------------------------
Mean value of the residuals -->  -0.017240717717302886

VARIANCE USING MATHEMATICAL FORMULA -->  0.0728
VARIANCE USING SCIKIT FUNCTION -->  0.0728
```

INFERENCE

> We see that the MSE 1% of MEDIAN imputation is the least, the value being -->
> **0.0071**

```
df_mean_stats
```

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	
...	...	...	
2408	0.194444	0.194444	
2409	-0.546296	-0.546296	
2410	-0.370370	-0.370370	
2411	-0.046296	-0.046296	
2412	-0.462963	-0.462963	

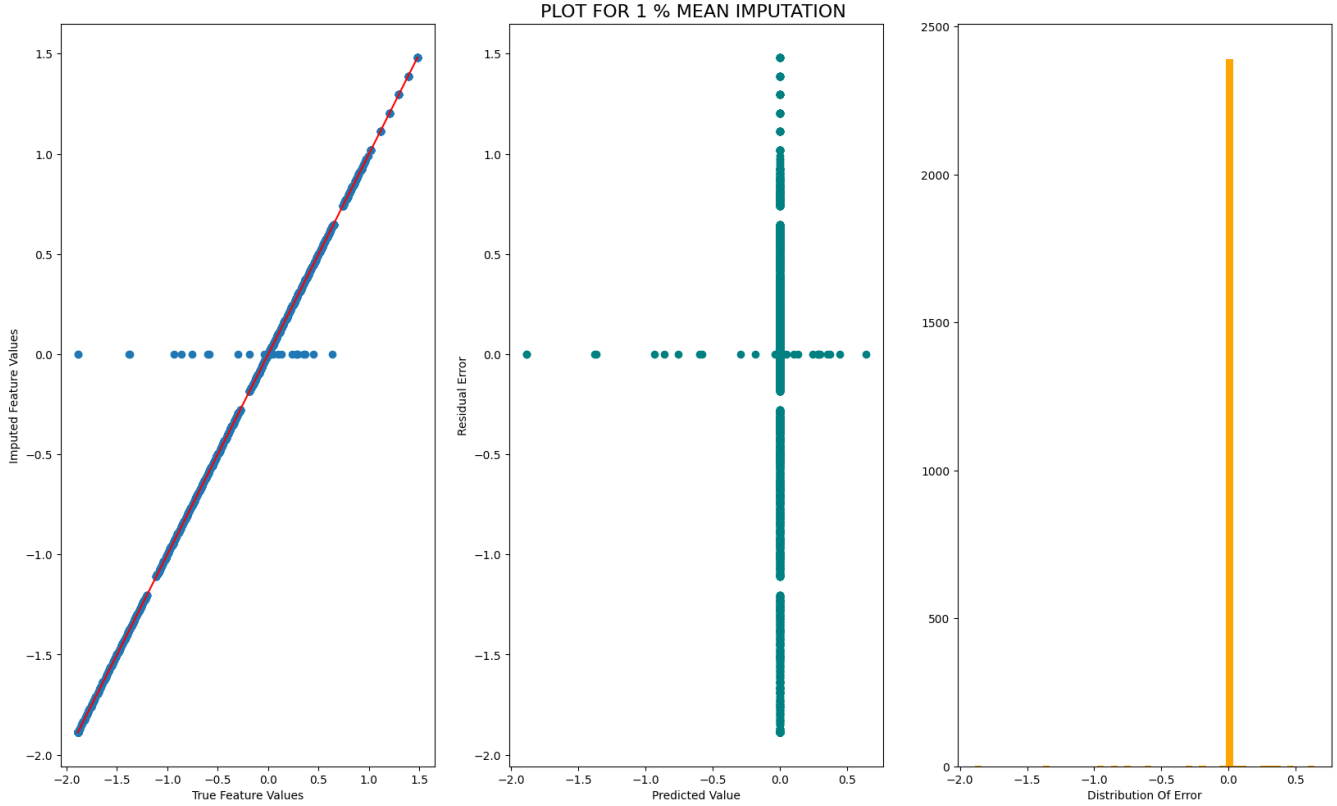
2413 rows × 13 columns

RESIDUAL PLOT FOR 1%, 5% AND 10% MEDIAN IMPUTATION

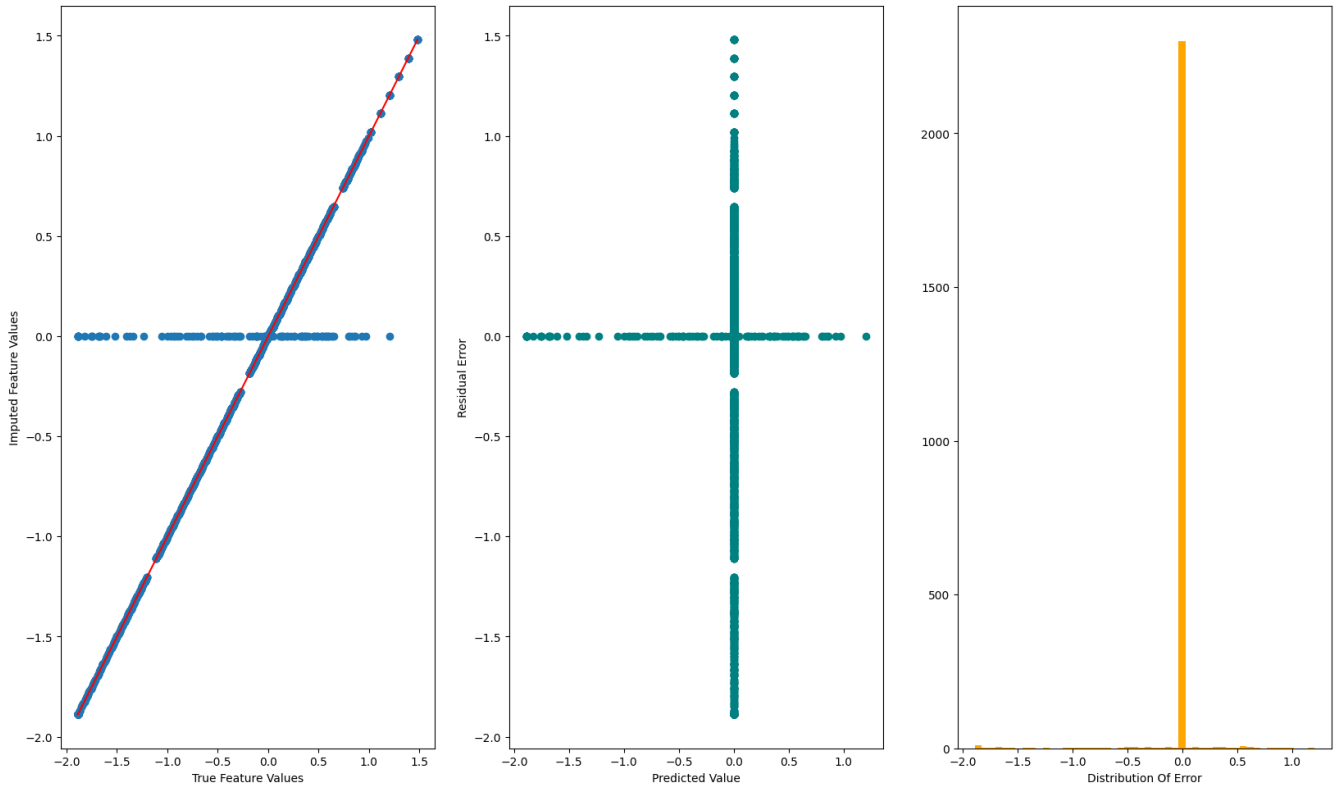
```
residual_plot(df_median_stats)
```

```
RESIDUAL PLOT FOR 1% IMPUTATION
RESIDUAL PLOT FOR 5% IMPUTATION
RESIDUAL PLOT FOR 10% IMPUTATION
```

PLOT FOR 1 % MEAN IMPUTATION



PLOT FOR 5 % MEAN IMPUTATION



INFERENCE

> We see that the Residual plots of all these graphs tell us that the difference between the actual and the imputed value is negligible and therefore we can choose this method for imputation for the current dataset.

IMPUTATION METHOD - MOST FREQUENT

Filling the Nan values with the **MOST FREQUENT** imputation

```
df_impute_mf = impute_values(df_fill_na, 'most_frequent')
df_impute_mf.head()
```

```
PERCENTAGE OF NULL VALUES BEFORE ADDING Nan
```

	column_name	percent_missing
winz_life_expectancy	winz_life_expectancy	0.0
winz_life_expectancy_percent_1	winz_life_expectancy_percent_1	0.0
winz_life_expectancy_percent_5	winz_life_expectancy_percent_5	0.0
winz_life_expectancy_percent_10	winz_life_expectancy_percent_10	0.0

	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy_pe
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	

```
df_mf_stats = statistical_measures(df_impute_mf)
```

```
MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2
```

MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:

$$\text{Residual Variance} = \frac{1}{(n-k-1)} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

==========
1% IMPUTATION
==========

MEAN SQUARE ERROR - MSE

MSE USING MATHEMATICAL FORMULA --> 0.02990225840028607
MSE USING SCIKIT FUNCTION --> 0.02990225840028607

VARIANCE

Mean value of the residuals --> 0.01564826326533745

VARIANCE USING MATHEMATICAL FORMULA --> 0.02966
VARIANCE USING SCIKIT FUNCTION --> 0.02967

==========
5% IMPUTATION
==========

MEAN SQUARE ERROR - MSE

MSE USING MATHEMATICAL FORMULA --> 0.15485358230481105
MSE USING SCIKIT FUNCTION --> 0.15485358230481105

VARIANCE

Mean value of the residuals --> 0.07710549339227335

VARIANCE USING MATHEMATICAL FORMULA --> 0.14891
VARIANCE USING SCIKIT FUNCTION --> 0.14897

==========
10% IMPUTATION
==========

MEAN SQUARE ERROR - MSE

MSE USING MATHEMATICAL FORMULA --> 0.3642963397850124
MSE USING SCIKIT FUNCTION --> 0.3642963397850124

VARIANCE

Mean value of the residuals --> 0.17141333210541668

VARIANCE USING MATHEMATICAL FORMULA --> 0.3349
VARIANCE USING SCIKIT FUNCTION --> 0.3351

INFERENCE

> We see that the MSE 1% of MOST FREQUENCY imputation is the least, the value being --> **0.032**

```
df_mf_stats
```

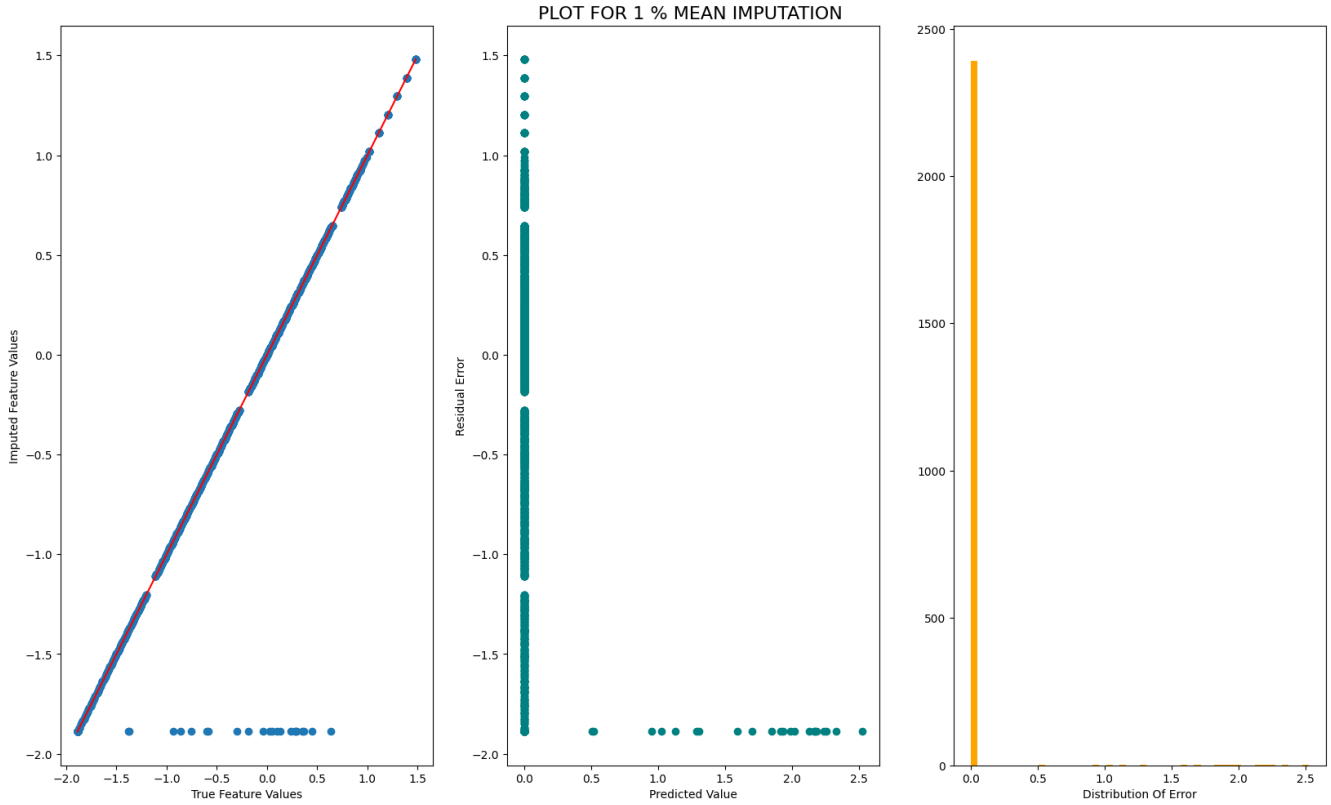
	winz_life_expectancy	winz_life_expectancy_percent_1	winz_life_expectancy
0	0.444444	0.444444	
1	0.240741	0.240741	
2	-1.888889	-1.888889	
3	0.314815	0.314815	
4	0.305556	0.305556	
...	...	...	
2408	0.194444	0.194444	
2409	-0.546296	-0.546296	
2410	-0.370370	-0.370370	
2411	-0.046296	-0.046296	
2412	-0.462963	-0.462963	

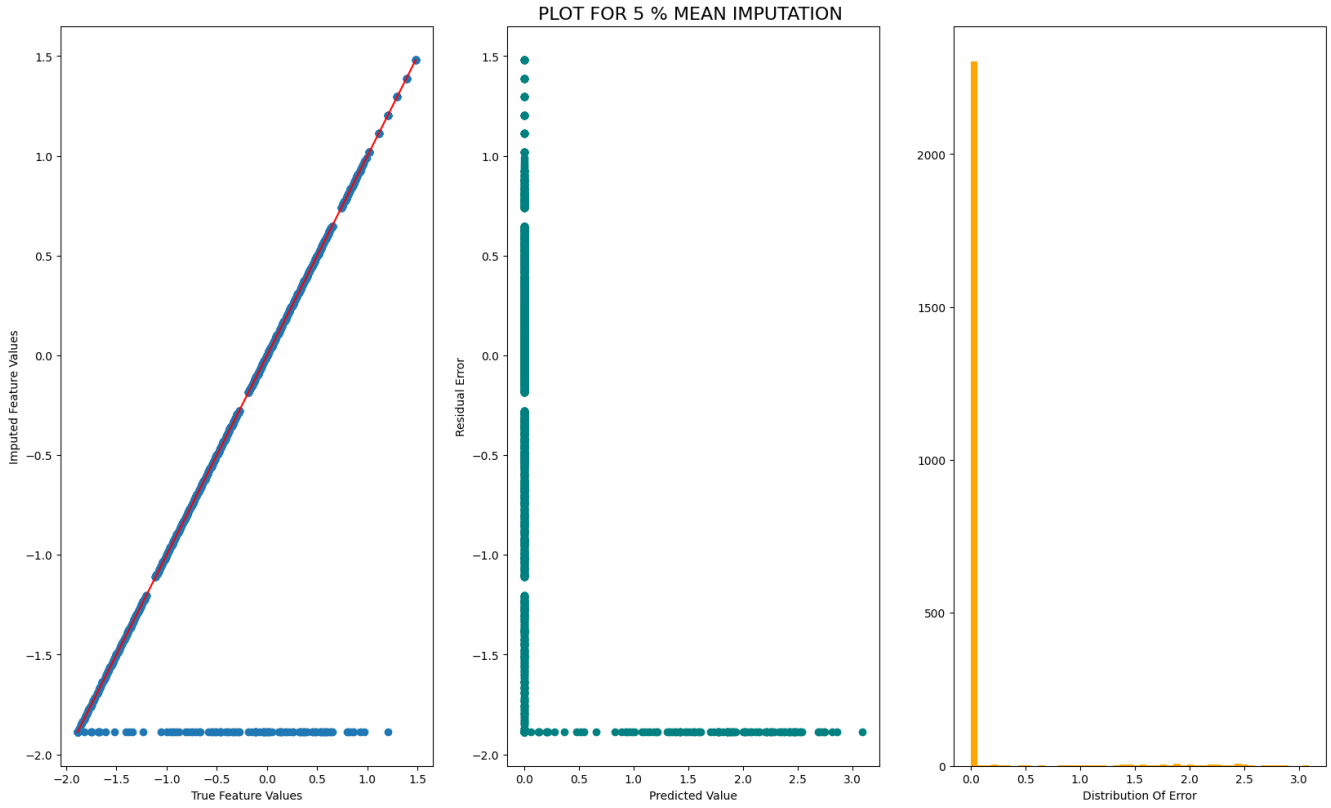
2413 rows × 13 columns

RESIDUAL PLOT FOR 1%, 5% AND 10% MEAN IMPUTATION

```
residual_plot(df_mf_stats)
```

```
RESIDUAL PLOT FOR 1% IMPUTATION
RESIDUAL PLOT FOR 5% IMPUTATION
RESIDUAL PLOT FOR 10% IMPUTATION
```





INFERENCE

> We see that the Residual plots of all these graphs tell us that the difference between the actual and the imputed value is negligible and therefore we can choose this method for imputation for the current dataset.

LIFE EXPECTANCY PREDICTION WITH H2O AUTOML

PROBLEM STATEMENT

> There are five main areas that can impact life expectancy:

- public health and medical care systems
- individual behaviors
- social and economic factors
- physical and social environments
- government policies and resource availability

> The aim of this project is to utilize information on mortality rates, socio-economic conditions, immunization status, and other health-related factors of a specific population.

> We will use the **H2O AutoML machine learning** tool to predict their life expectancy. Additionally, we will also employ various **visualization techniques** such as:

- Variable Importance Plot
- Partial Dependence Plot
- Individual Conditional
- Expectation Plot
- SHAP Summary Plot

  to elucidate how each feature input influences our model's prediction.

WHAT IS AutoML?

> **Automated Machine Learning (AutoML)** is the process of automating tasks in the machine learning pipeline such as
>
> 1. data preprocessing
> 2. hyperparameter tuning
> 3. model selection and evaluation

> H2O AutoML trains and cross validates the following models:
>
> 1. Three pre-specified XGBoost GBM (Gradient Boosting Machine) models
> 2. Fixed grid of GLMs (Generalized Linear Model)
> 3. Default Random Forest (DRF)
> 4. 5 pre-specified H2O GBMs
> 5. Near-default Deep Neural Net
> 6. Extremely Randomized Forest (XRT)
> 7. Random grid of XGBoost GBMs
> 8. Random grid of H2O GBMs
> 9. Random grid of Deep Neural Nets
> 10. Multiple Stacked Ensemble models will also be trained throughout the process

REFERENCES

https://towardsdatascience.com/automated-machine-learning-with-h2o-258a2f3a203f

https://medium.com/mlearning-ai/life-expectancy-prediction-with-h2o-automl-91a36b4b06d2

EXPLORATORY DATA ANALYSIS (EDA) : AUTO ML

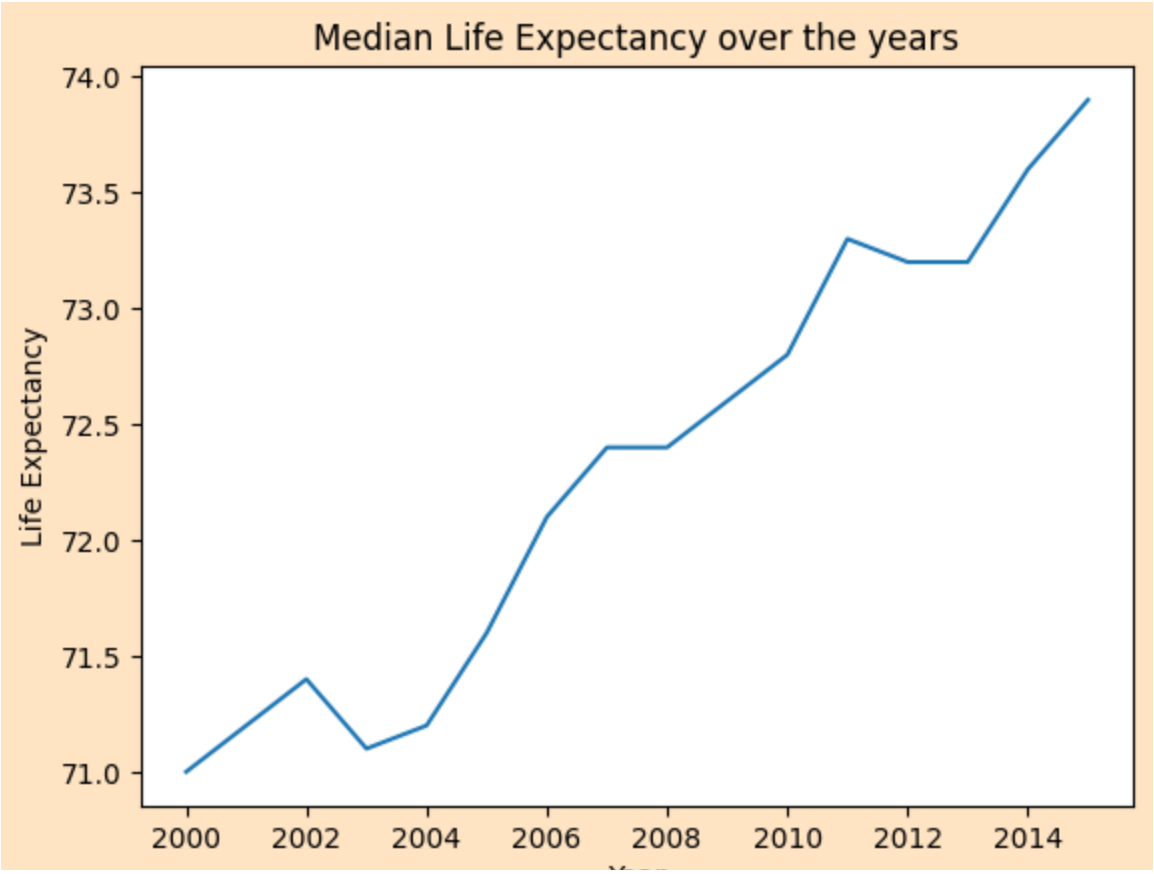
Analysing Median Life Expectancy over the years

```
#Finding the median life_expectancy values over the years to analyse the trend of lif

df_median_life_expectancy = df_original[['year','life_expectancy']]
df_median_life_expectancy = df_median_life_expectancy.groupby('year',as_index=False).
df_median_life_expectancy
```

	year	life_expectancy
0	2000	71.0
1	2001	71.2
2	2002	71.4
3	2003	71.1
4	2004	71.2
5	2005	71.6
6	2006	72.1
7	2007	72.4
8	2008	72.4
9	2009	72.6
10	2010	72.8
11	2011	73.3
12	2012	73.2
13	2013	73.2
14	2014	73.6
15	2015	73.9

```
#ploting line chart using pyplot
plt.figure(figsize=(20,10))

#adding the baxkground color
plt.figure(facecolor= 'bisque')

#plotting x and y axis
plt.plot(df_median_life_expectancy['year'],df_median_life_expectancy['life_expectancy

plt.xlabel('Year')

plt.ylabel('Life Expectancy')
plt.title('Median Life Expectancy over the years')




plt.show()
```

```
<Figure size 2000x1000 with 0 Axes>
```



INFERENCE

> From the above graph we can see that `life_expectancy` is increasing over the years. This has been mainly possible due to the improvement in the healthcare and medical facilities and also due to the increase in the resources and budget allocated by the government.

PERFORMING ONE-HOT ENCODING ON THE CATEGORICAL DATA COLUMN : STATUS

```
df_encoder_original = df_outlier_original

status = pd.get_dummies(df_encoder_original['status'])
#status1 = pd.get_dummies(df_encoder_original['country'])
#status2 = pd.get_dummies(df_encoder_original['year'])
#.drop('Developing',axis=1)
df_encoder_original = pd.concat([df_encoder_original,status], axis=1)
#df_encoder_original = pd.concat([df_encoder_original,status1], axis=1)
```

```
#df_encoder_original = pd.concat([df_encoder_original,status2], axis=1)
display(df_encoder_original)
```

	country	year	status	life_expectancy	adult_mortality	infant_deaths
16	Albania	2015	Developing	77.8	74.0	0
32	Algeria	2015	Developing	75.6	19.0	21
48	Angola	2015	Developing	52.4	335.0	66
64	Antigua and Barbuda	2015	Developing	76.4	13.0	0
80	Argentina	2015	Developing	76.3	116.0	8
...	...	...	...	...	...	...
2825	Uruguay	2000	Developing	75.1	131.0	1
2841	Uzbekistan	2000	Developing	67.1	189.0	30
2857	Vanuatu	2000	Developing	69.0	18.0	0
2873	Venezuela (Bolivarian Republic of)	2000	Developing	72.5	168.0	11
2905	Yemen	2000	Developing	68.0	252.0	48

2413 rows × 38 columns

DATA VISUALIZATION WITH DATAPREP

---

1. Dataprep can be used to create an automatic data report.
2. Dataprep automatically detects schemas, data types, possible joins, and anomalies such as missing values, outliers, and duplicates.
3. This saves time and effort in assessing data quality.
4. To create the report, we simply pass our dataframe to the create_report function.
5. The report includes information on
    1. dataset statistics

> 2. insights
> 3. variables
> 4. interactions
> 5. correlations
> 6. missing values

```
df_encoder_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 38 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   country                               2413 non-null   object 
 1   year                                  2413 non-null   int64  
 2   status                                2413 non-null   object 
 3   life_expectancy                       2413 non-null   float64
 4   adult_mortality                       2413 non-null   float64
 5   infant_deaths                         2413 non-null   int64  
 6   alcohol                               2413 non-null   float64
 7   percentage_expenditure                2413 non-null   float64
 8   hepatitis_b                           2413 non-null   float64
 9   measles                               2413 non-null   int64  
 10  under-five_deaths                     2413 non-null   int64  
 11  polio                                 2413 non-null   float64
 12  total_expenditure                     2413 non-null   float64
 13  diphtheria                            2413 non-null   float64
 14  hiv/aids                              2413 non-null   float64
 15  gdp                                   2413 non-null   float64
 16  population                            2413 non-null   float64
 17  thinness_1-19_years                   2413 non-null   float64
 18  thinness_5-9_years                    2413 non-null   float64
 19  income_composition_of_resources       2413 non-null   float64
 20  schooling                             2413 non-null   float64
 21  log_percentage_expenditure            2413 non-null   float64
 22  log_population                        2413 non-null   float64
 23  log_gdp                               2413 non-null   float64
 24  winz_life_expectancy                  2413 non-null   float64
 25  winz_adult_mortality                  2413 non-null   float64
 26  winz_alcohol                          2413 non-null   float64
 27  winz_hepatitis_b                      2413 non-null   float64
 28  winz_polio                            2413 non-null   float64
 29  winz_total_expenditure                2413 non-null   float64
 30  winz_diphtheria                       2413 non-null   float64
 31  winz_income_composition_of_resources  2413 non-null   float64
 32  winz_schooling                        2413 non-null   float64
 33  winz_hiv/aids                         2413 non-null   float64
 34  winz_thinness_1-19_years              2413 non-null   float64
 35  winz_thinness_5-9_years               2413 non-null   float64
```

```
 36  Developed                        2413 non-null   uint8
 37  Developing                       2413 non-null   uint8
dtypes: float64(30), int64(4), object(2), uint8(2)
memory usage: 702.2+ KB
```

CREATING A REPORT

```
#create report

df_life_expectancy = df_encoder_original.drop(['country','status'], axis=1)
report = create_report(df_life_expectancy, title = "Life Expectancy Report")
report
```

Life Expectancy Report | Overview | Variables ≡ | Interactions | Correlations

Missing Values

Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	664.5 KB
Average Row Size in Memory	282.0 B
Variable Types	Numerical: 33 Categorical: 3

`infant_deaths` and `under-five_deaths` have similar distributions	Similar Distribution
`alcohol` and `winz_alcohol` have similar distributions	Similar Distribution
`polio` and `diphtheria` have similar distributions	Similar Distribution
`total_expenditure` and `winz_total_expenditure` have similar distributions	Similar Distribution
`thinness_1-19_years` and `thinness_5-9_years` have similar distributions	Similar Distribution
`thinness_1-19_years` and `winz_thinness_1-19_years` have similar distributions	Similar Distribution
`thinness_1-19_years` and `winz_thinness_5-9_years` have similar distributions	Similar Distribution
`thinness_5-9_years` and `winz_thinness_1-19_years` have similar distributions	Similar Distribution

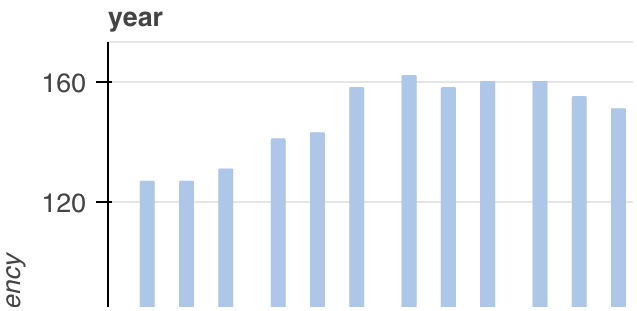
1 2 3 4

Variables

Sort by Feature order ☐ Reverse order

Approximate Distinct Count	16
Approximate Unique (%)	0.7%

Size	
Mean	2007.8036
Minimum	2000
Maximum	2015



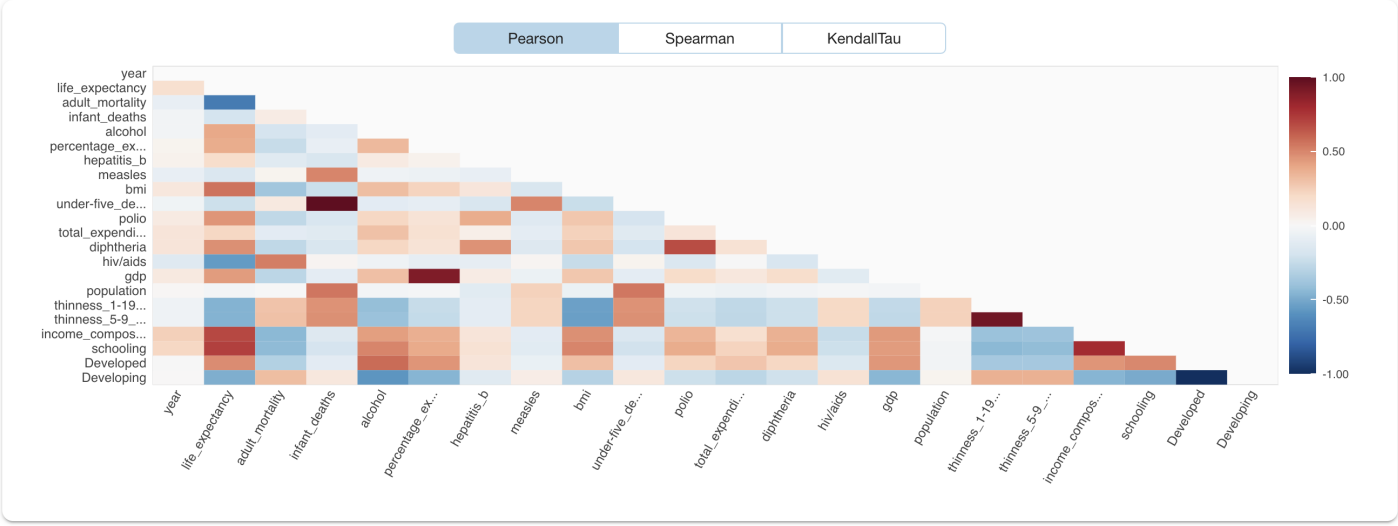
Missing 0 Zeros 0

SUMMARIZATION OF THE REPORT

Infinite (%) 0.0% Negatives

CORRELATION

Correlations



INFERENCE

From the above Correlation result we can conclude that there are 4 pairs of highly correlated variables

1. `infant_deaths` and `under-five_deaths`
2. `percentage_expenditure` and `gdp`
3. `income_composition_of_resources` and `schooling`
4. `thinness_1_to_19_years` and `thinness_5_to_9_years`

Positive Correlations with `life_expectancy`

1. `gdp`
2. `total_expenditure`
3. `percentage_expenditure`
4. `income_composition_of_resources`

5. `status_Developed`
6. `schooling`
7. `bmi`
8. `polio vaccine`
9. `diphtheria vaccine`
10. `hepatitis_b vaccine`
11. `alcohol`

Negative Correlations with `life_expectancy`

1. `status_developing`
2. `under-five_deaths`
3. `adult_mortality`
4. `infant_deaths`
5. `thinness_1_to_19_years`
6. `thinness_5_to_9_years`
7. `hiv/aids`
8. `measle`

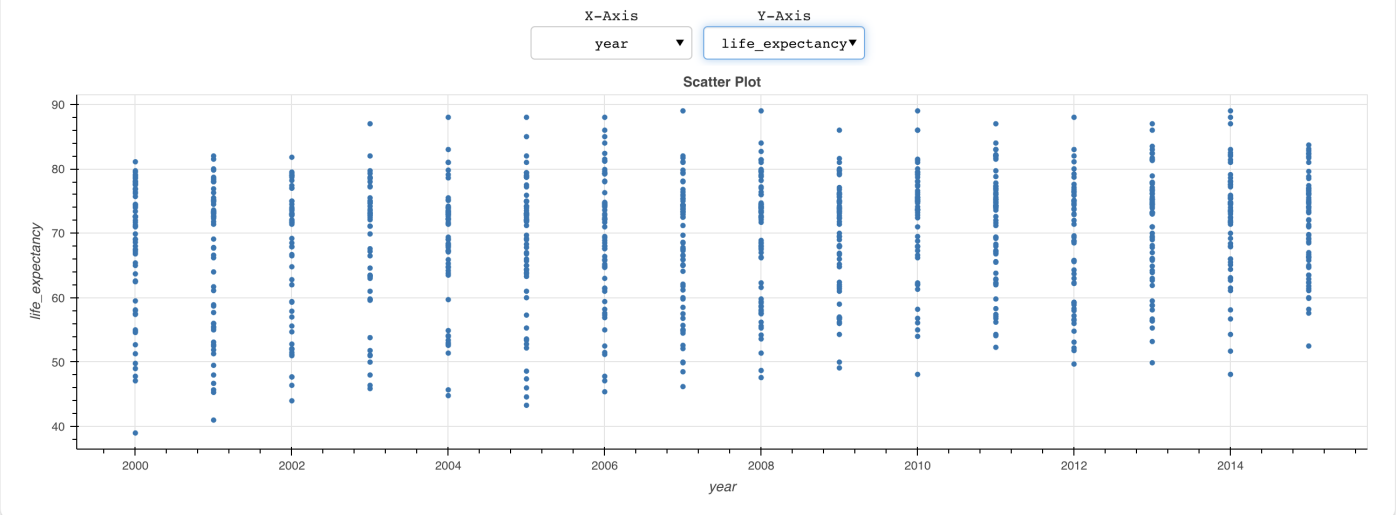
Missing 0 Zeros 0

DETRMINING THE RELATIONSHIP BETWEEN `life_expectancy` AND OTHER INDEPENDENT VARIABLES

Negatives 0.0%

1. `year` and `life_expectancy`

Interactions



Infinite (%) 0.0%

INFERENCE

`year`

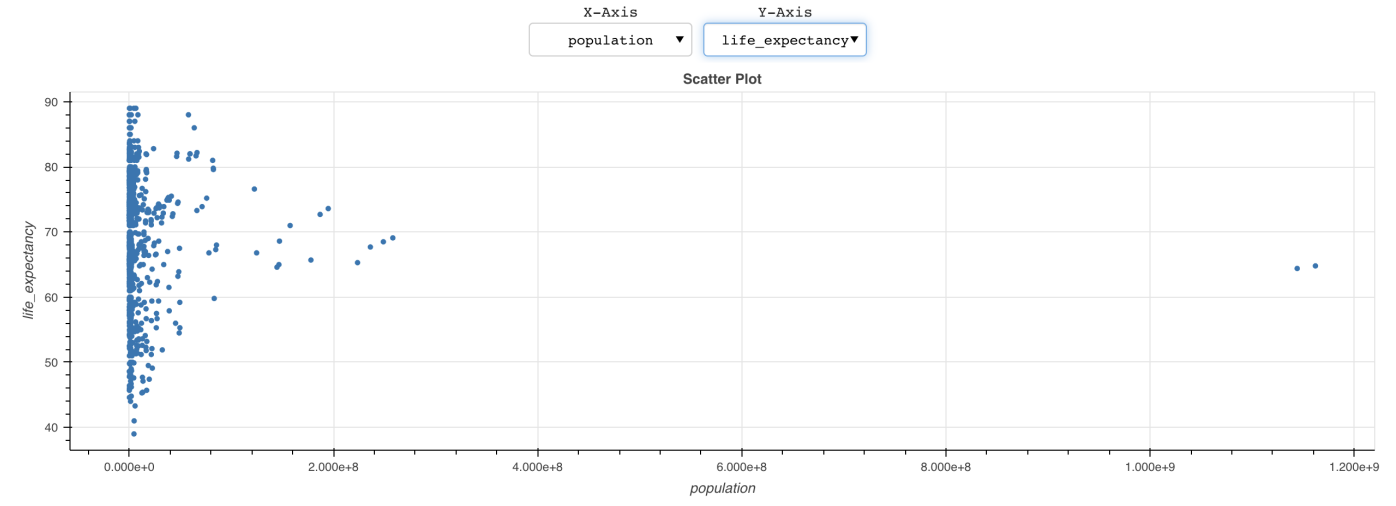
1. As the year passes, we see an improvement (increase) in the `life-expectancy`
2. This is mainly due to the improvement in the medical and health sector and the resource and fund allocation provided by the government.

Approximate Minimum 0

2. `population` and `life_expectancy`

Zeros

Interactions

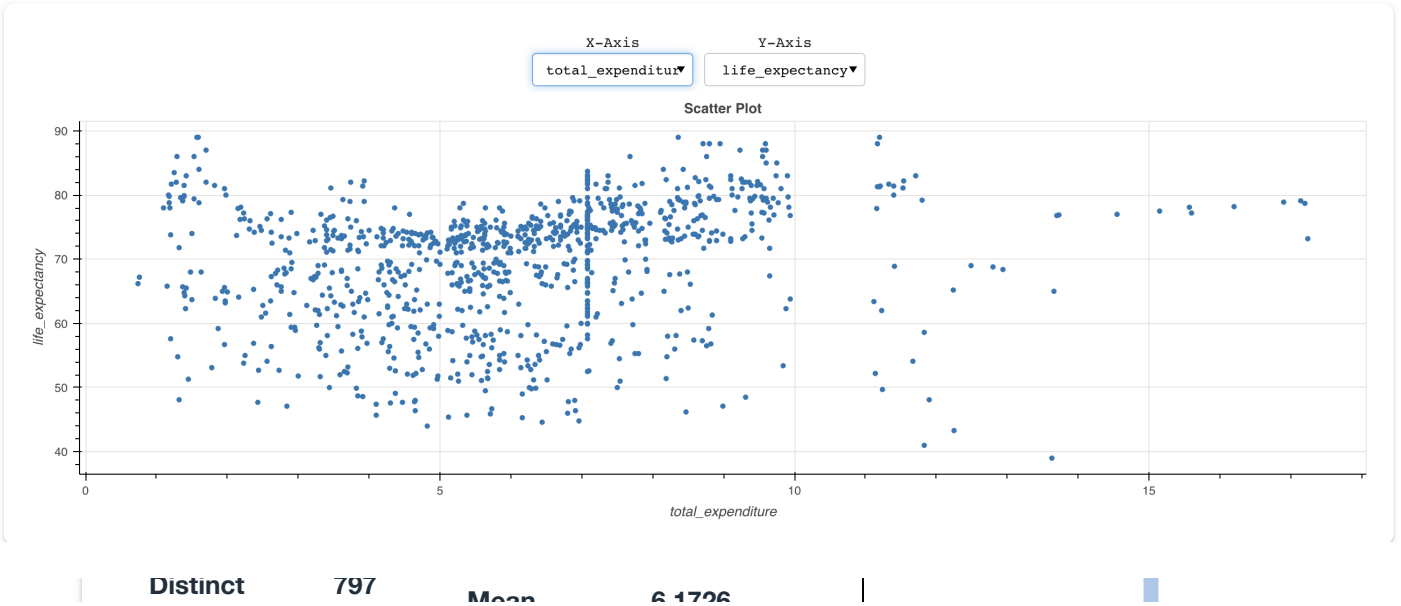


INFERENCE

`population`

1. We see that as the population increases, the `life_expectancy` of people increases, mainly as the tendancy of death among the infants and adults is rare and the older ones live longer.
2. `population` --> **positively correlated** with `life_expectancy`

3. `total_expenditure` and `life_expectancy`



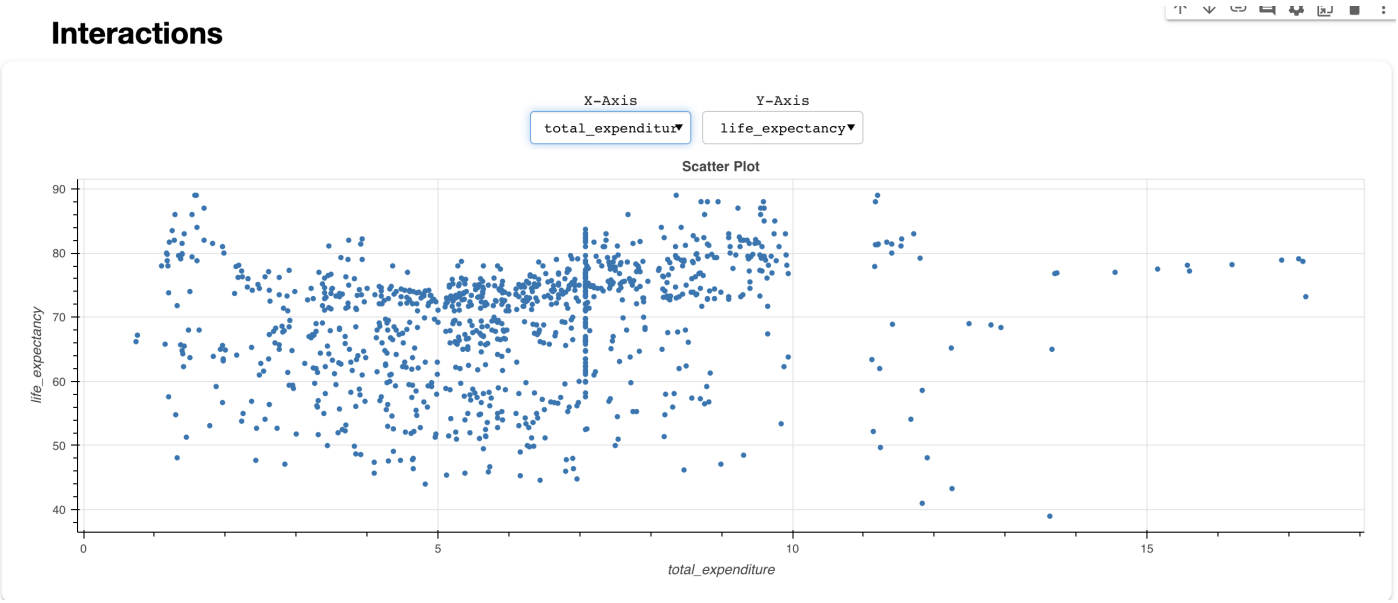
INFERENCE

`population`

1. We see that as the population increases, the `life_expectancy` of people increases, mainly as the tendancy of death among the infants and adults is rare and the older ones live longer.
2. `population` --> **positively correlated** with `life_expectancy`

3. `total_expenditure` and `life_expectancy`

Interactions



INFERENCE

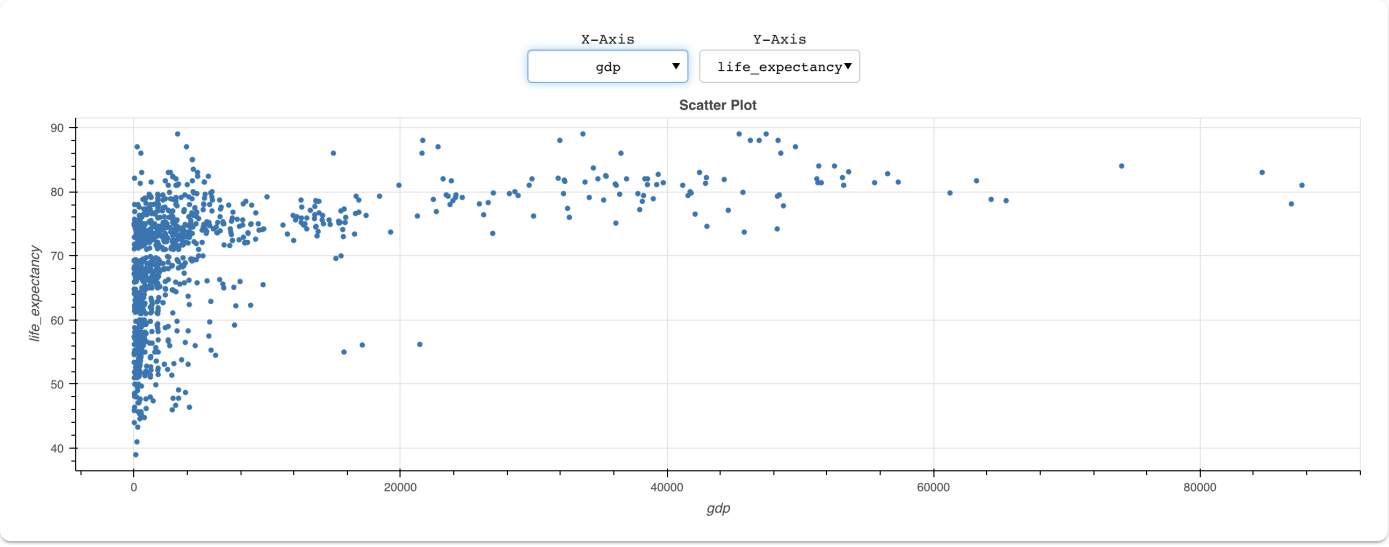
`total_expenditure`

1. We see that as the total_expenditure invested for healthcare by the government increases , the `life_expectancy` of people increases, mainly due to better standards in the healthcare sector.
2. `total_expenditure` --> **positively correlated** with `life_expectancy`

Infinite (%) **0.0%**

4. `gdp` and `life_expectancy`

Interactions



nfinite (%) 0.0%

INFERENCE

`gdp`

1. GDP serves as an indicator of both economic progress and the quality of life. Studies have shown that improved economic conditions and increased spending on healthcare are linked to higher life expectancy.
2. `gdp` --> **positively correlated** with `life_expectancy`

Unique (%) **Maximum** **19.7** cy

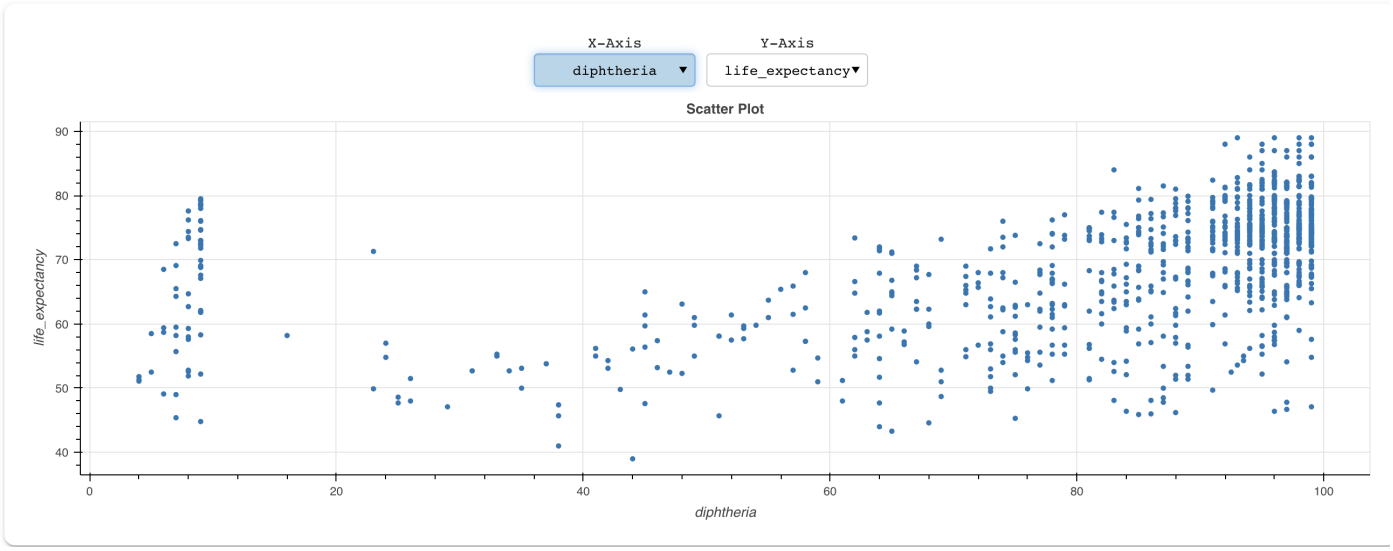
5. `polio` and `life_expectancy`

   `measles` and `life_expectancy`

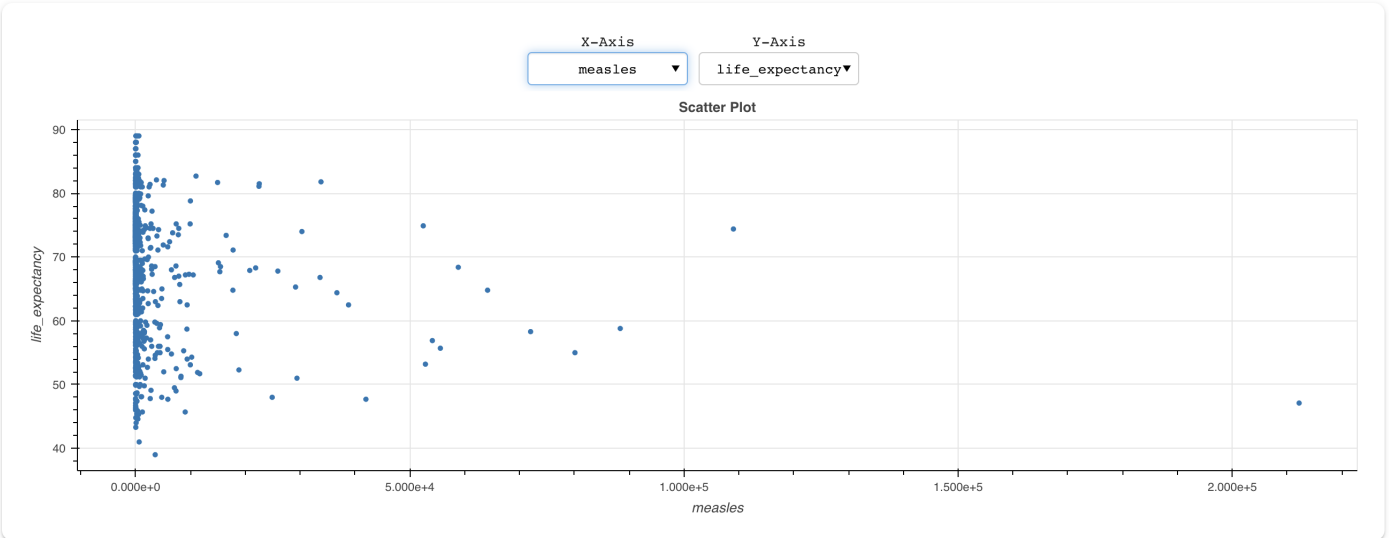
   `diphtheria` and `life_expectancy`

Negatives (%) **0.0%**

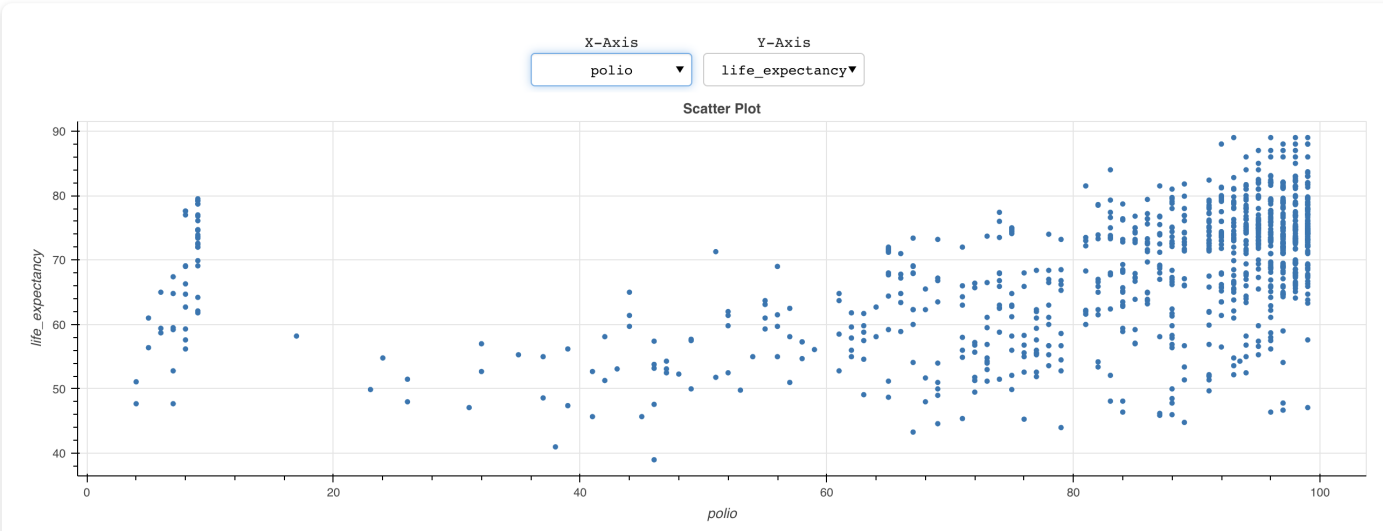
Interactions



Interactions



Interactions



INFERENCE

`polio` , `measles`, `diptheria` --> Immunization Factors

1. Vaccination and immunization against diseases plays a crucial factor is increasing the `life_expectancy` of people.
2. `polio`, `measles`, `diptheria` --> **positively correlated** with `life_expectancy`

	Infinite (%)	**0.0%**	..	..

AutoML H2O

> 1. H2O AutoML is an automated machine learning tool provided by H2O.ai, which aims to simplify and streamline the process of developing machine learning models.
> 2. It automates many of the time-consuming and complex tasks involved in model selection, hyperparameter tuning, feature engineering, and ensembling.
> 3. This allows users with limited expertise in machine learning to quickly and easily build high-performing models.
> 4. H2O AutoML supports a wide range of algorithms, including deep learning, gradient boosting, and generalized linear models, and can be used for a variety of tasks such as regression, classification, and time-series forecasting.

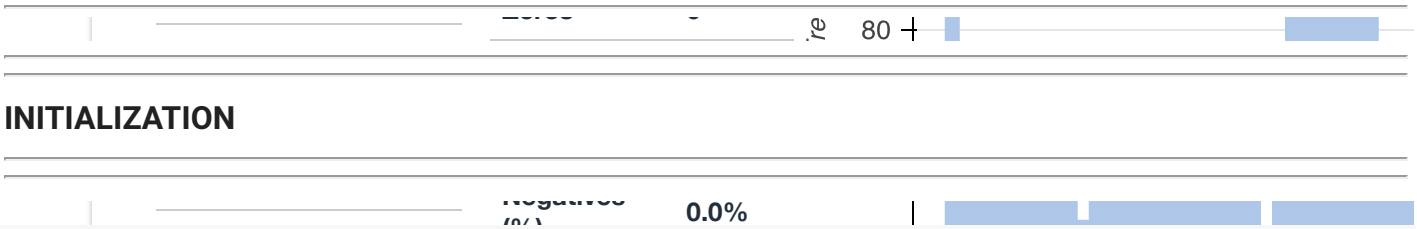
> **FEATURES OF AutoML**

> 1.**Hyperparameter tuning**: One of the key features of H2O AutoML is its ability to automate the process of hyperparameter tuning. Hyperparameters are settings that must be chosen prior to model training, and they can have a significant impact on model performance. H2O AutoML uses advanced optimization algorithms to automatically search for the best combination of hyperparameters for each algorithm and dataset.

2. **Feature engineering**: H2O AutoML automates feature engineering, which involves creating new features from the existing data that can improve model performance. It uses a range of techniques, such as one-hot encoding, feature scaling, and missing value imputation, to create new features that can improve model accuracy.

3. **Highly Flexibe** : H2O AutoML is designed to be highly flexible and can be used for a wide range of machine learning tasks, including classification, regression, and time-series forecasting. It supports a variety of popular algorithms, such as XGBoost, Random Forest, and Deep Learning, and can handle large datasets with millions of rows and thousands of columns.

> Overall, H2O AutoML is a powerful tool that can help data scientists, machine learning engineers, and business analysts to quickly build high-performing models without requiring advanced machine learning expertise.



INITIALIZATION


```
#initiate
h2o.init()
```

```
Checking whether there is an H2O instance running at http://localhost:54321.....
```

CONVERT Pandas df INTO H2O FRAME

> The h2o.H2OFrame serves as the equivalent of a dataframe for H2O, similar to how pandas provides dataframes for Python.

```
#convert pandas df into h2o frame
h2o_df = h2o.H2OFrame(df_life_expectancy)
```

```
Parse progress: |
```

```
#preview
h2o_df
```

year	life_expectancy	adult_mortality	infant_deaths	alcohol	percentage_expend
2015	77.8	74	0	4.6	3
2015	75.6	19	21	5.285	
2015	52.4	335	66	5.285	
2015	76.4	13	0	5.285	
2015	76.3	116	8	5.285	
2015	74.8	118	1	5.285	
2015	82.8	59	1	5.285	
2015	81.5	65	0	5.285	
2015	72.7	118	5	5.285	
2015	76.1	147	0	5.285	

[2413 rows x 36 columns]

```
#statistics
h2o_df.describe()
```

```
Rows:2413
Cols:36
```

	year	life_expectancy	adult_mortality	infant_deaths	
type	int	real	int	int	
mins	2000.0	36.3	1.0	0.0	
mean	2007.80356402818	70.51193535018629	155.2436800663075	10.33733941152093	4.915
maxs	2015.0	89.0	723.0	549.0	
sigma	4.524618917340964	8.970740078426696	119.3523372054428	24.40062091174184	3.94
zeros	0	0	0	843	
missing	0	0	0	0	
0	2015.0	77.8	74.0	0.0	
1	2015.0	75.6	19.0	21.0	
2	2015.0	52.4	335.0	66.0	
3	2015.0	76.4	13.0	0.0	
4	2015.0	76.3	116.0	8.0	
5	2015.0	74.8	118.0	1.0	
6	2015.0	82.8	59.0	1.0	
7	2015.0	81.5	65.0	0.0	
8	2015.0	72.7	118.0	5.0	
9	2015.0	76.1	147.0	0.0	

TRAIN TEST SPLIT

1. split the dataframe to get train and test datasets using *split_frame*
2. Considered ratios=[.75] to split THE data into 75% for training and 25% for validation.
3. X will be all of our columns that are taken as independent variables.
4. y is the dependent variable `life_expectancy`.

```
#split train and test sets
train, test = h2o_df.split_frame(ratios=[.75])
```

```
#Define X - independent variables and Y - dependent variable (life_expectancy)

X = train.columns
y = 'life_expectancy'
X.remove(y)
```

Independent Variables: They can also be referred as Features

```
X
```

```
['year',
 'adult_mortality',
 'infant_deaths',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'measles',
 'under-five_deaths',
 'polio',
 'total_expenditure',
 'diphtheria',
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',
 'schooling',
 'log_percentage_expenditure',
 'log_population',
 'log_gdp',
 'winz_life_expectancy',
 'winz_adult_mortality',
 'winz_alcohol',
 'winz_hepatitis_b',
 'winz_polio',
 'winz_total_expenditure',
 'winz_diphtheria',
 'winz_income_composition_of_resources',
 'winz_schooling',
 'winz_hiv/aids',
 'winz_thinness_1-19_years',
 'winz_thinness_5-9_years',
```

```
 'Developed',
 'Developing']
```

Independent Variable --> `life_expectancy` : This is our Target

```
y
```

```
'life_expectancy'
```

PERFORMING RIDGE AND LASSO REGULARIZATION

> Ridge and Lasso Regression are two regularization techniques commonly used in machine learning to prevent overfitting in linear regression models.

> In Ridge Regression, a penalty term is added to the cost function that includes the squared sum of the magnitude of the coefficients. This encourages the coefficients to be small, which can help prevent overfitting.

$$\text{minimize} \quad \left[\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2\right] \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq t$$

> In Lasso Regression, a penalty term is added to the cost function that includes the absolute sum of the magnitude of the coefficients. This encourages some of the coefficients to become zero, effectively performing feature selection and simplifying the model.

$$\text{LASSO:} \ \min_{\beta} \frac{1}{2n} \sum_{i=1}^{n}(y_i - \boldsymbol{X_i}\boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

> Both Ridge and Lasso Regression can be tuned with a hyperparameter that controls the strength of the regularization. The optimal value of this hyperparameter can be determined using techniques such as cross-validation.

> Overall, Ridge and Lasso Regression are powerful tools for improving the generalization performance of linear regression models, particularly in high-dimensional feature spaces.

Developing Approximate Unique (%) 0.1%

```
df_life_expectancy[X]
```

	year	adult_mortality	infant_deaths	alcohol	percentage_expenditure	hepa
16	2015	74.0	0	4.600	364.975229	
32	2015	19.0	21	5.285	0.000000	
48	2015	335.0	66	5.285	0.000000	
64	2015	13.0	0	5.285	0.000000	
80	2015	116.0	8	5.285	0.000000	
...	...	...	...	...	...	
2825	2000	131.0	1	6.650	645.958382	
2841	2000	189.0	30	1.600	48.509417	
2857	2000	18.0	0	1.210	21.900752	
2873	2000	168.0	11	8.010	0.000000	
2905	2000	252.0	48	0.070	0.000000	

2413 rows × 35 columns

```
from sklearn import linear_model


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_life_expectancy[X], df_life_ex


X1 = df_model_original[columns]
y1 = df_model_original['winz_life_expectancy']


X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size = 0.3, rand
```

```
# L1 Regularization using Lasso Regression
lasso = linear_model.Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# L2 Regularization using Ridge Regression
ridge = linear_model.Ridge(alpha=0.1)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

#print('RMSE: %.2f' % cv_results['test-rmse-mean'].min())

print('L1 Regularization (Lasso) RMSE: %.2f' %lasso_rmse)
print('L2 Regularization (Ridge) RMSE: %.2f' %ridge_rmse)
```

```
L1 Regularization (Lasso) RMSE: 0.79
L2 Regularization (Ridge) RMSE: 0.78
Ill-conditioned matrix (rcond=1.90879e-19): result may not be accurate.
```

TRAIN

> H2O AutoML interface is designed to have as few parameters as possible so that all we need to do is to specify the dataset.

1. **aml:** is our automatic machine learning models.
2. **H2OAutoML:** When we run H2OAutoML, it will train various models as listed in the Methodology section.
3. **max_model:** train a maximum of 10 models, excluding the Stacked Ensemble models.
4. **stopping_metric**='RMSE' is the value used for early stopping so that we don't overfit our models.
5. **seed**=121 to ensure reproducibility.

> train by fitting in X and y while passing our train set with training_frame=train.

```
#initiate
aml = H2OAutoML(stopping_metric='RMSE',     #for regression
                seed=121,
                max_models=10)

#train
aml.train(x=X,
```

```
        y=y,
        training_frame=train)
```

```
AutoML progress: |████████████████████████████████████████████████████████████
```

```
Model Details
=============
H2OStackedEnsembleEstimator : Stacked Ensemble
Model Key: StackedEnsemble_AllModels_1_AutoML_1_20230424_132641
```

Model Summary for Stacked Ensemble:

key	value
Stacking strategy	cross_validation
Number of base models (used / total)	7/10
# GBM base models (used / total)	3/4
# XGBoost base models (used / total)	3/3
# GLM base models (used / total)	1/1
# DRF base models (used / total)	0/2
Metalearner algorithm	GLM
Metalearner fold assignment scheme	Random
Metalearner nfolds	5
Metalearner fold_column	None
Custom metalearner hyperparameters	None

MODEL EVALUATION

> 2 reports are generated by `H2OAutoML` on train data and cross-validation data

> The smaller the value of the metrics - `rmse`, `mse`, `mae`, `rmsle`, `deviance`, the better the model fits.

> `R^2`: is the coefficient of determination. The higher the value of `R^2`, the higher quality the model is i.e. how the model predicts to new data (validation set): is it overfiting or underfitting.

> An **AIC score** is a number used to determine which machine learning model is best for a given data set in situations where one can't easily test a data set. An AIC test is most useful when you're working with a small data set or time series analysis. The lower the AIC score the better.

```
RMSLE: 0.009942002612413551
```

TRAIN SET

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on train data. **

MSE: 0.3219671816291827
RMSE: 0.5674215202379821
MAE: 0.3610898046348926 5
RMSLE: 0.008150270026032245
Mean Residual Deviance: 0.3219671816291827
R^2: 0.9964247298478924
Null degrees of freedom: 2218
Residual degrees of freedom: 2205
Null deviance: 199829.70394959775
Residual deviance: 714.4451760351564
AIC: 3812.4439529565025
```

CROSS-VALIDATION SET

---

1. In machine learning, cross-validation is a technique used to evaluate how well a machine learning model generalizes to new, unseen data.
2. Cross-validation involves splitting the dataset into multiple subsets, called folds. The model is then trained on a portion of the data (e.g., using four of the five folds) and tested on the remaining fold. This process is repeated multiple times, with each fold serving as the validation set once.
3. The results from each fold are averaged together to provide an estimate of how well the model will perform on new, unseen data. By using cross-validation, we can get a more accurate and reliable estimate of the model's performance than by simply evaluating it on a single test set.

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on cross-validation data. **

MSE: 3.1258948564784617
RMSE: 1.768020038483292
MAE: 1.131596675969015
RMSLE: 0.026634304448066498
Mean Residual Deviance: 3.1258948564784617
R^2: 0.9652886405302595
Null degrees of freedom: 2218
Residual degrees of freedom: 2207
Null deviance: 200533.1580132012
Residual deviance: 6936.360686525706
AIC: 8852.289213485401
```

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
mae	1.1288439	0.0318190	1.1136681	1.1285526	1.1659096	1.084786	1.1513034
mean_residual_deviance	3.1277785	0.1986257	3.4059734	3.1384912	3.0379994	2.866327	3.1901023
mse	3.1277785	0.1986257	3.4059734	3.1384912	3.0379994	2.866327	3.1901023
null_deviance	40106.633	2505.41	41056.07	40188.71	42131.68	35800.938	41355.758
r2	0.9649896	0.0019408	0.9623743	0.965649	0.967544	0.9639528	0.9654277
residual_deviance	1388.922	105.0806660	1539.5	1374.659	1345.8337	1255.4512	1429.1658
rmse	1.7678403	0.0561196	1.845528	1.7715787	1.7429857	1.6930231	1.7860857
rmsle	0.0266234	0.0013983	0.0268463	0.0266319	0.0285049	0.0245673	0.0265664

LEADERBOARD EXPLORATION

---

1. The aml object has a dashboard called the leaderboard that shows all the models trained in the process, along with their scores and metrics for ranking the models. For example, in our multivariate regression case, the metric used is MRSE.

2. The leaderboard displays the models with their corresponding metrics. When given an H2OAutoML object, the leaderboard shows 5-fold cross-validated metrics by default.

```
#leaderboard
lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

model_id	rmse	mse	mae	
StackedEnsemble_AllModels_1_AutoML_1_20230424_132641	0.507873	0.257935	0.200096	0
GBM_2_AutoML_1_20230424_132641	0.528828	0.279659	0.216283	0.
StackedEnsemble_BestOfFamily_1_AutoML_1_20230424_132641	0.531277	0.282256	0.221401	0.
GBM_3_AutoML_1_20230424_132641	0.538695	0.290192	0.262118	0.
GBM_4_AutoML_1_20230424_132641	0.548674	0.301043	0.264492	0.
XGBoost_3_AutoML_1_20230424_132641	0.648671	0.420774	0.355686	0.
GLM_1_AutoML_1_20230424_132641	0.687662	0.472879	0.281417	0.
XGBoost_2_AutoML_1_20230424_132641	0.711397	0.506086	0.422837	0.
XRT_1_AutoML_1_20230424_132641	0.76331	0.582642	0.406069	0.
DRF_1_AutoML_1_20230424_132641	0.77198	0.595953	0.408536	0.
XGBoost_1_AutoML_1_20230424_132641	0.84293	0.710532	0.498984	0

INFERENCE

1. The models are ranked based on their performance measures, including `mean_residual_deviance`, `rmse`, `mse`, `mae`, and `rmsle`.
2. The models with smaller error values are considered to be a better fit and have higher performance.
3. The top-performing model is
    1. **StackedEnsemble_AllModels_1_AutoML_1_20230225_03829**
    2. second-best model, **StackedEnsemble_BestOfFamily_1_AutoML_1_20230225_03829**
4. The All Models ensemble includes all the models, while the Best of Family ensemble includes the best-performing model from each algorithm class or family.

MODEL EXPLAINABILITY IN AUTOML H2O

1. H2O offers ways to explain how machine learning models work for both groups of models and individual models.

2. Model explainability is vital in machine learning because evaluating a model's accuracy alone is not enough to understand how the model makes decisions.
3. We must also comprehend how the model input variables function and how the model's predictions are affected when we alter the input variable values.

When h2o.explain() is provided with a list of models, the following global explanations will be generated by default:

1. Residual Analysis for Leader Model
2. Variable Importance of Top Base (non-Stacked) Model
3. Variable Importance Heatmap (compare all non-Stacked models)
4. Model Correlation Heatmap (compare all models)
5. SHAP Summary of Top Tree-based Model (TreeSHAP)
6. Partial Dependence (PD) Multi Plots (compare all models)
7. Individual Conditional Expectation (ICE) Plots

```
#compare all models
exm = aml.explain(test)
```

Leaderboard

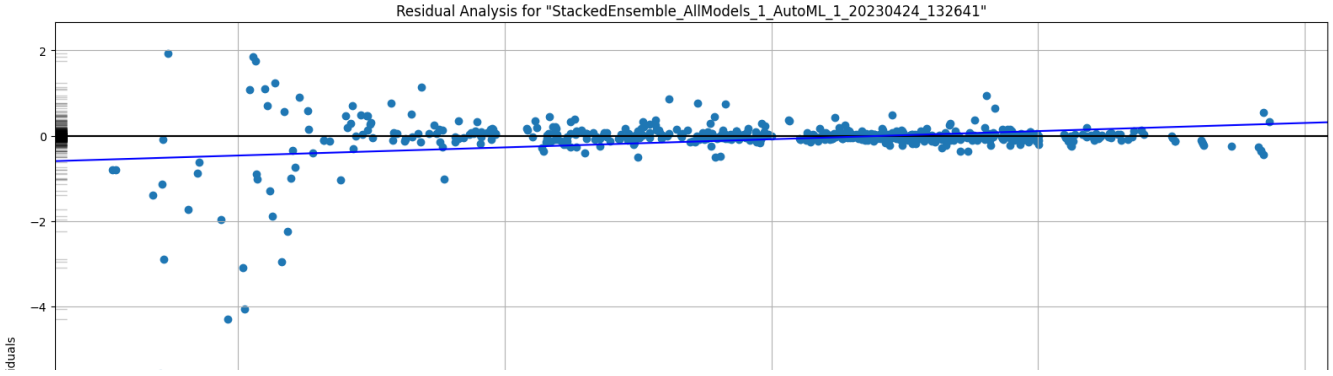
Leaderboard shows models with their metrics. When provided with H2OAutoML object, the leaderboard shows 5-fold cross-validated metrics by default (depending on the H2OAutoML settings), otherwise it shows metrics computed on the frame. At most 20 models are shown by default.

model_id	rmse	mse	mae	
XGBoost_1_AutoML_1_20230424_132641	0.73813	0.544836	0.367784	0.
GBM_3_AutoML_1_20230424_132641	0.747487	0.558737	0.269384	0.0
XGBoost_3_AutoML_1_20230424_132641	0.750857	0.563786	0.298774	0.0
StackedEnsemble_AllModels_1_AutoML_1_20230424_132641	0.77297	0.597482	0.218718	0.0
StackedEnsemble_BestOfFamily_1_AutoML_1_20230424_132641	0.807726	0.652422	0.239782	0.0
GBM_2_AutoML_1_20230424_132641	0.817075	0.667611	0.237494	0.0
GBM_4_AutoML_1_20230424_132641	0.830131	0.689117	0.312168	0.0
XGBoost_2_AutoML_1_20230424_132641	0.848503	0.719957	0.3899	0.0
DRF_1_AutoML_1_20230424_132641	1.01071	1.02154	0.436461	0.
XRT_1_AutoML_1_20230424_132641	1.02093	1.0423	0.434431	0.0
GLM_1_AutoML_1_20230424_132641	1.03142	1.06384	0.352414	0.0
GBM_1_AutoML_1_20230424_132641	1.22761	1.50703	0.536027	0.0

[12 rows x 9 columns]

Residual Analysis

Residual Analysis plots the fitted values vs residuals on a test dataset. Ideally, residuals should be randomly distributed. Patterns in this plot can indicate potential problems with the model selection, e.g., using simpler model than necessary, not accounting for heteroscedasticity, autocorrelation, etc. Note that if you see "striped" lines of residuals, that is an artifact of having an integer valued (vs a real valued) response variable.

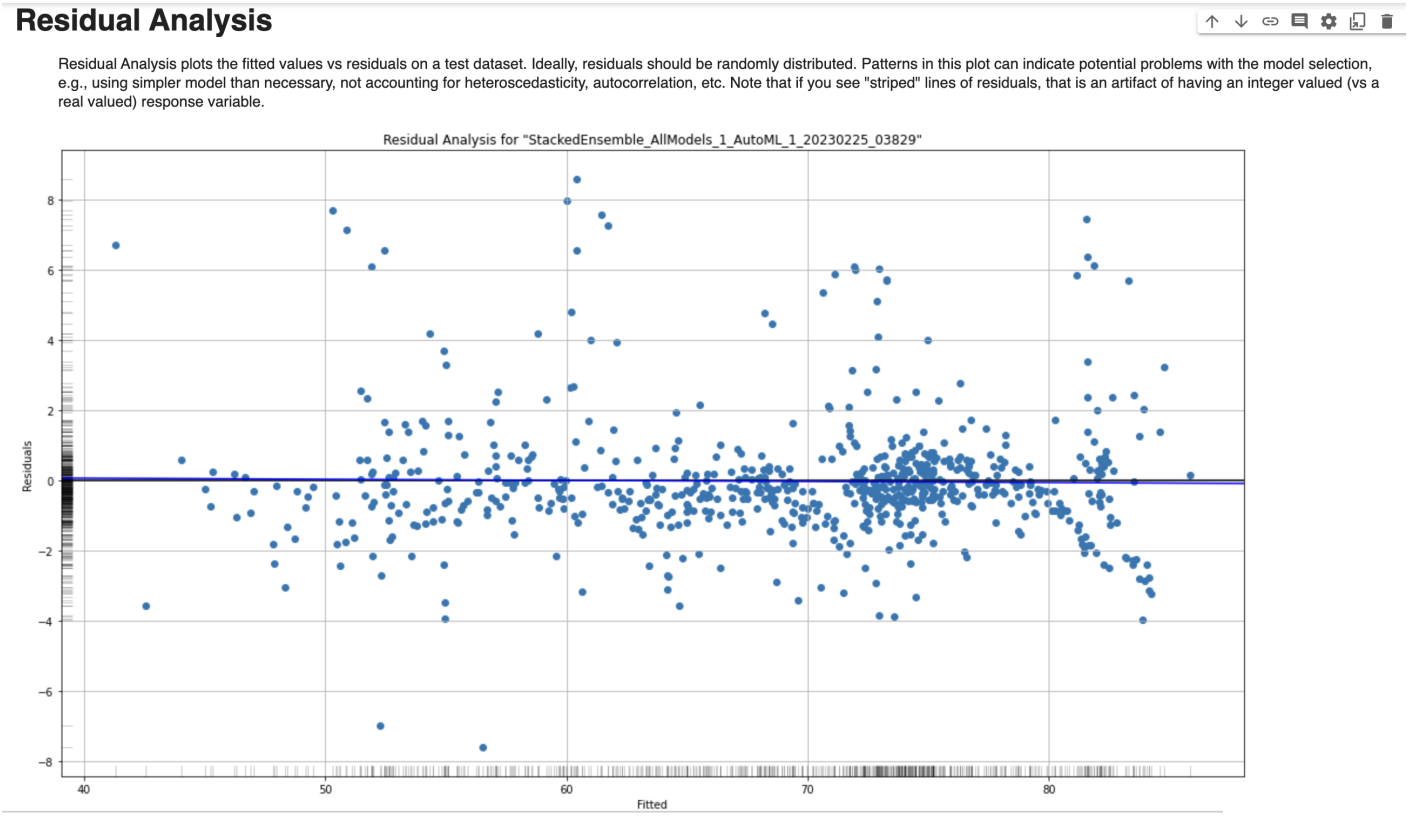


SUMMARIZATION OF THE OBSERVATIONS

1. RESIDUAL ANALYSIS

> It is a technique to check if the model predictions are accurate by analyzing the difference between the predicted and actual values of the target variable. The analysis helps in identifying patterns or outliers in the errors, which can be used to further improve the model.

for tree-based algorithms. This plot can be useful for determining whether the model overfits.



EXPLANATION : RESIDUAL ANALYSIS

> 1. **Residual** is the difference between the actual value of the dependent variable (y) and the predicted value (ŷ).

2. In H2O, we can see the residuals plotted on the y-axis and the actual values on the x-axis. By looking at this plot, we can tell how far off our predictions are from the actual values.

3. If the data points are scattered randomly around the horizontal line, it means that the residuals are just random errors and the linear regression model we chose is suitable for our data.

---

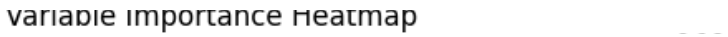
2. VARIABLE IMPORTANCE OF TOP BASE (NON-STACK) MODEL

---

1. It helps in understanding which variables (features) are the most important for making predictions in the top performing non-Stacked model. It can help in identifying the key drivers of the target variable and can also be used for feature selection.

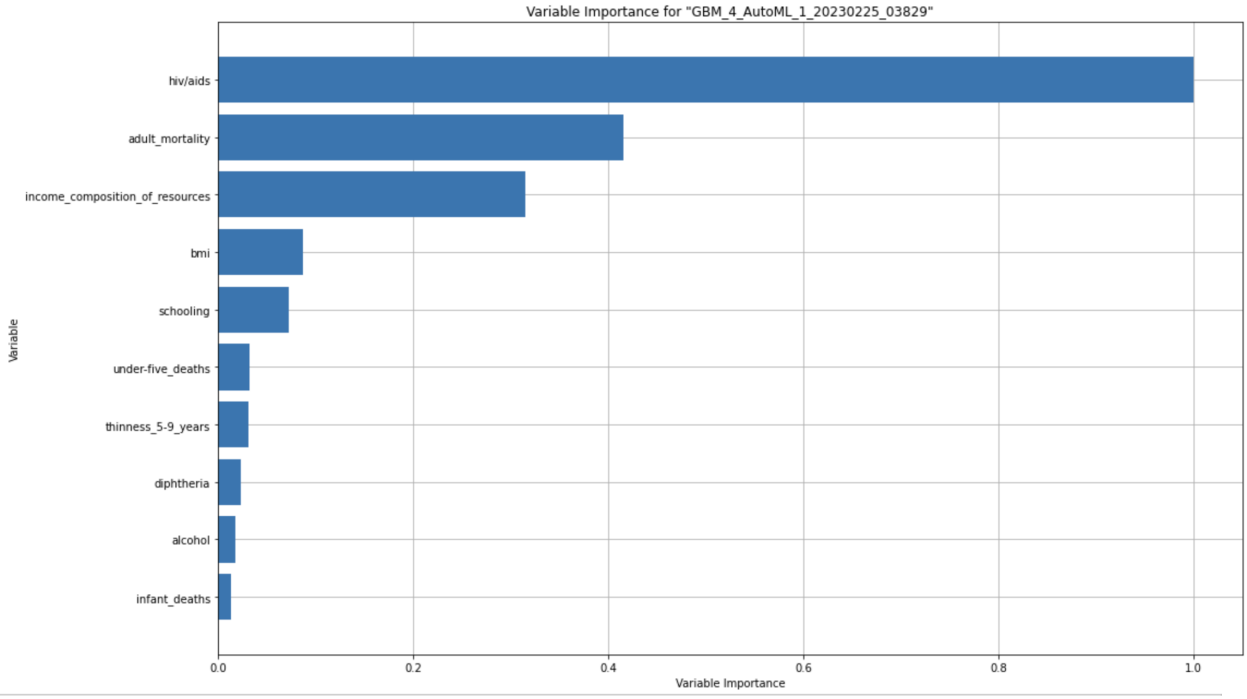
2. In the context of model explainability, "non-stacked" refers to the models that were trained individually without being combined in an ensemble model.

3. In machine learning, ensemble refers to a technique of combining multiple models to improve their predictive performance. The idea is that by combining several weaker models, we can create a more robust and accurate model.



Variable Importance

The variable importance plot shows the relative importance of the most important variables in the model.



This plot shows the correlation between the predictions of the models. For classification

EXPLANATION : VARIABLE IMPORTANCE OF TOP BASE (NON-STACK) MODEL

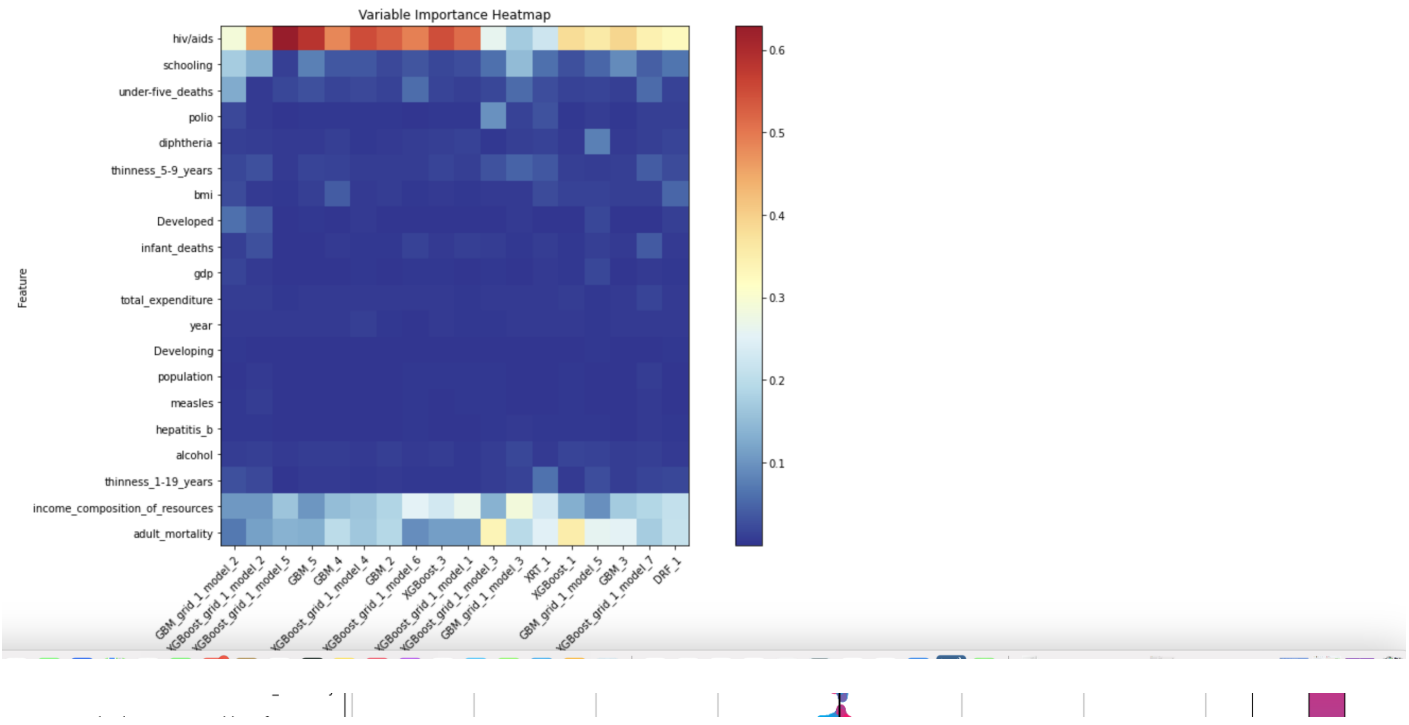
> The features `hiv/aids`, `income_composition_of_resources`, and `adult_mortality` are the ones that play the biggest role in the performance of the GBM_4 model, which is one of the best-performing models. This means that they have the most impact on the predictions made by the model

3. VARIABLE IMPORTANCE HEATMAP (COMPARE ALL NON-STACK MODELS) MODEL

> It is a graphical representation that compares the variable importance of all non-Stacked models in the AutoML process. It can help in identifying the most important variables across all models and can be used for feature selection.

Variable Importance Heatmap

Variable importance heatmap shows variable importance across multiple models. Some models in H2O return variable importance for one-hot (binary indicator) encoded versions of categorical columns (e.g. Deep Learning, XGBoost). In order for the variable importance of categorical columns to be compared across all model types we compute a summarization of the the variable importance across all one-hot encoded features and return a single variable importance for the original categorical feature. By default, the models and variables are ordered by their similarity.



EXPLANATION : VARIABLE IMPORTANCE HEATMAP (COMPARE ALL NON-STACK MODELS) MODEL

1. `Hiv/aids` is the most significant feature in the majority of models including `XGBoost_grid_1_model_2`, `XGBoost_grid_1_model_5`, `GBM_5`, `GBM_4`, `XGBoost_grid_1_model_4`, `GBM_2`, `XGBoost_grid_1_model_6`, `XGBoost_3`,`XGBoost_grid_1_model_4`.
2. Income composition of resources is important in 2 out of 10 models, including `XGBoost_grid_1_model_6`, `XGBoost_grid_1_model_3`.
3. Adult mortality is also a critical factor in 2 out of 10 models, including `XGBoost_grid_1_model_3`, `XGBoost_3` .

Partial Dependence Plots

4. MODEL CORRELATION

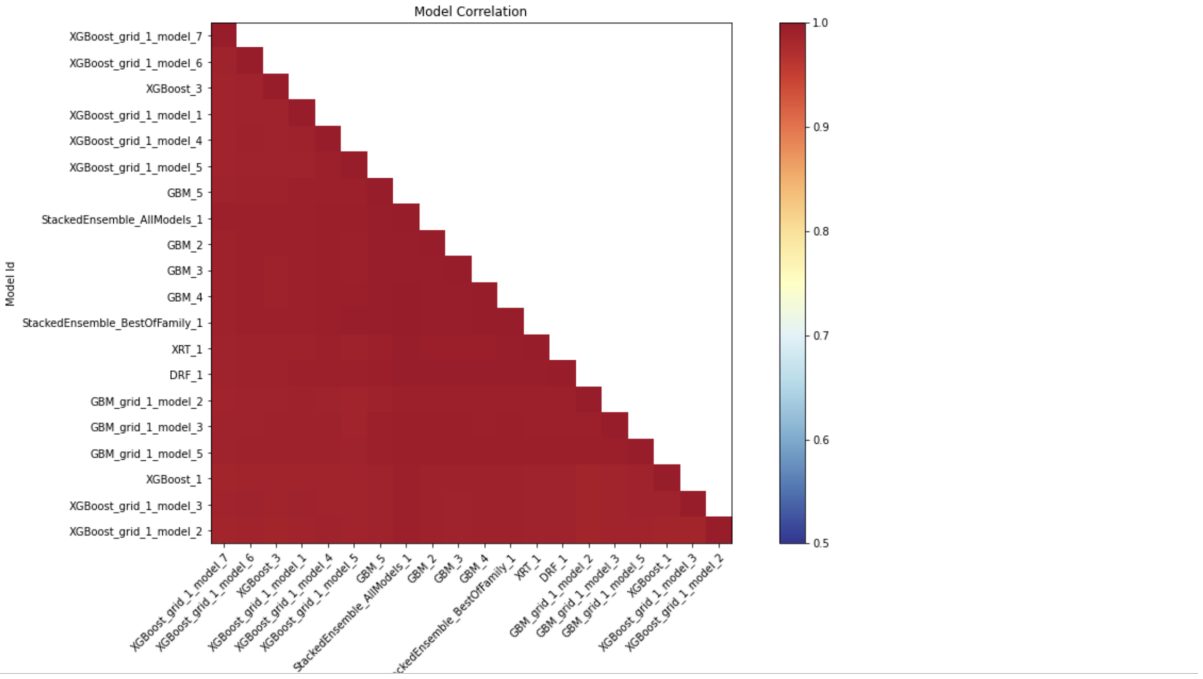
It is a graphical representation that compares the correlation between all models in the AutoML process. It can help in identifying the models that are highly correlated

and can be used for model selection.

80

Model Correlation

This plot shows the correlation between the predictions of the models. For classification, frequency of identical predictions is used. By default, models are ordered by their similarity (as computed by hierarchical clustering). Interpretable models, such as GAM, GLM, and RuleFit are highlighted using red colored text.



EXPLANATION : MODEL CORRELATION

All these models are highly correlated with each other and produce similar predictions.

5. SHAP SUMMARY

1. The SHAP Summary of the Top Tree-based Model (TreeSHAP) is a tool used to explain how a specific Tree-based model makes individual predictions. It summarizes the most important variables that affect the model's predictions for each observation in the dataset.

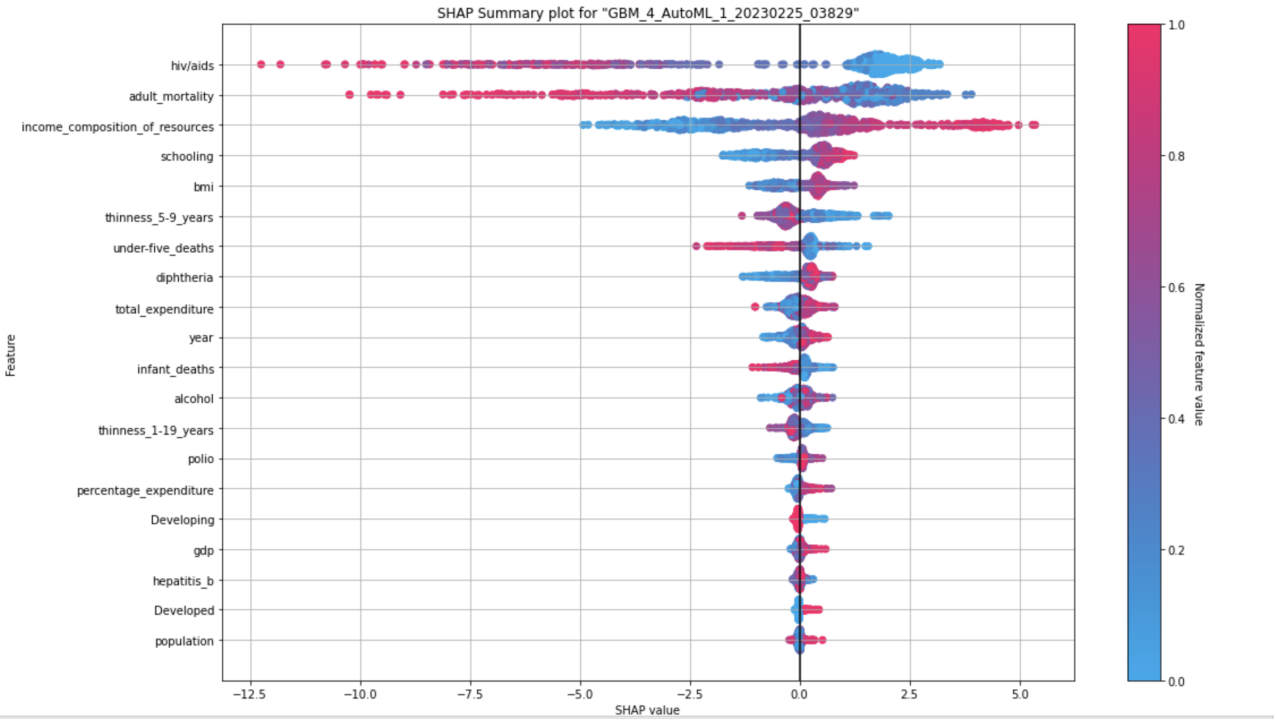
2. Tree-based models are a type of machine learning model that uses decision trees to make predictions.

3. In a decision tree, the model is represented as a tree-like structure, where each node represents a decision based on the input features. The tree is constructed by splitting the data into smaller subsets based on the values of the input features, and recursively constructing smaller decision trees for each subset.

4. Tree-based models are often used for classification and regression tasks and can handle both numerical and categorical input features. Examples of tree-based models include Random Forest, Gradient Boosted Trees, and Decision Trees.

SHAP Summary

SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.



Partial Dependence plot for "winz_income_composition_of_resources"

EXPLANATION : SHAP SUMMARY

---

1. According to the Variable Importance Plot, some features have a stronger impact on the output than others.

2. `Hiv/aids`, `adult_mortality`, `income_composition_of_resources`, and `schooling` are the most important features in determining the outcome.

3. High values of `Hiv/aids` and `adult_mortality` have a **negative** impact on the outcome, while high values of `income_composition_of_resources` and `schooling` have a **positive** impact.

4. Other features such as `thinness`, `BMI`, `under-five_deaths`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.

---

6. PARTIAL DEPENDENCE PLOTS (PDP)

1. It is a graphical representation that compares the partial dependence of all models in the AutoML process.
2. It helps in understanding the relationship between the target variable and the key variables across all models.

EXPLANATION : PARTIAL DEPENDENCE PLOTS (PDP)

1. According to the Variable Importance Plot, some features have a stronger impact on the output than others.

2. `Hiv/aids`, `adult_mortality`, `income_composition_of_resources`, and `schooling` are the most important features in determining the outcome.

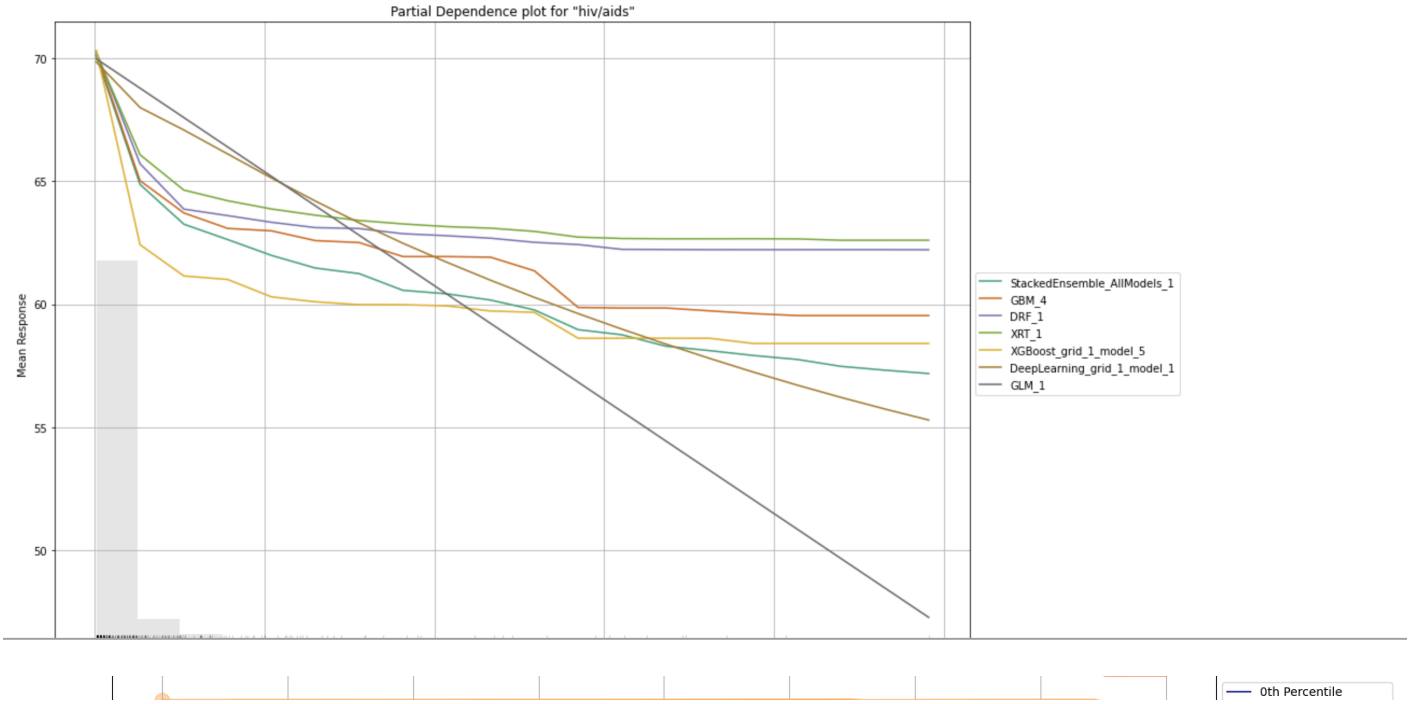
3. High values of `Hiv/aids` and `adult_mortality` have a **negative** impact on the outcome, while high values of `income_composition_of_resources` and `schooling` have a **positive** impact.

4. Other features such as `thinness`, `BMI`, `under-five_deaths`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.

OBSERVATION 1 : `hiv/aids`

Partial Dependence Plots

Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

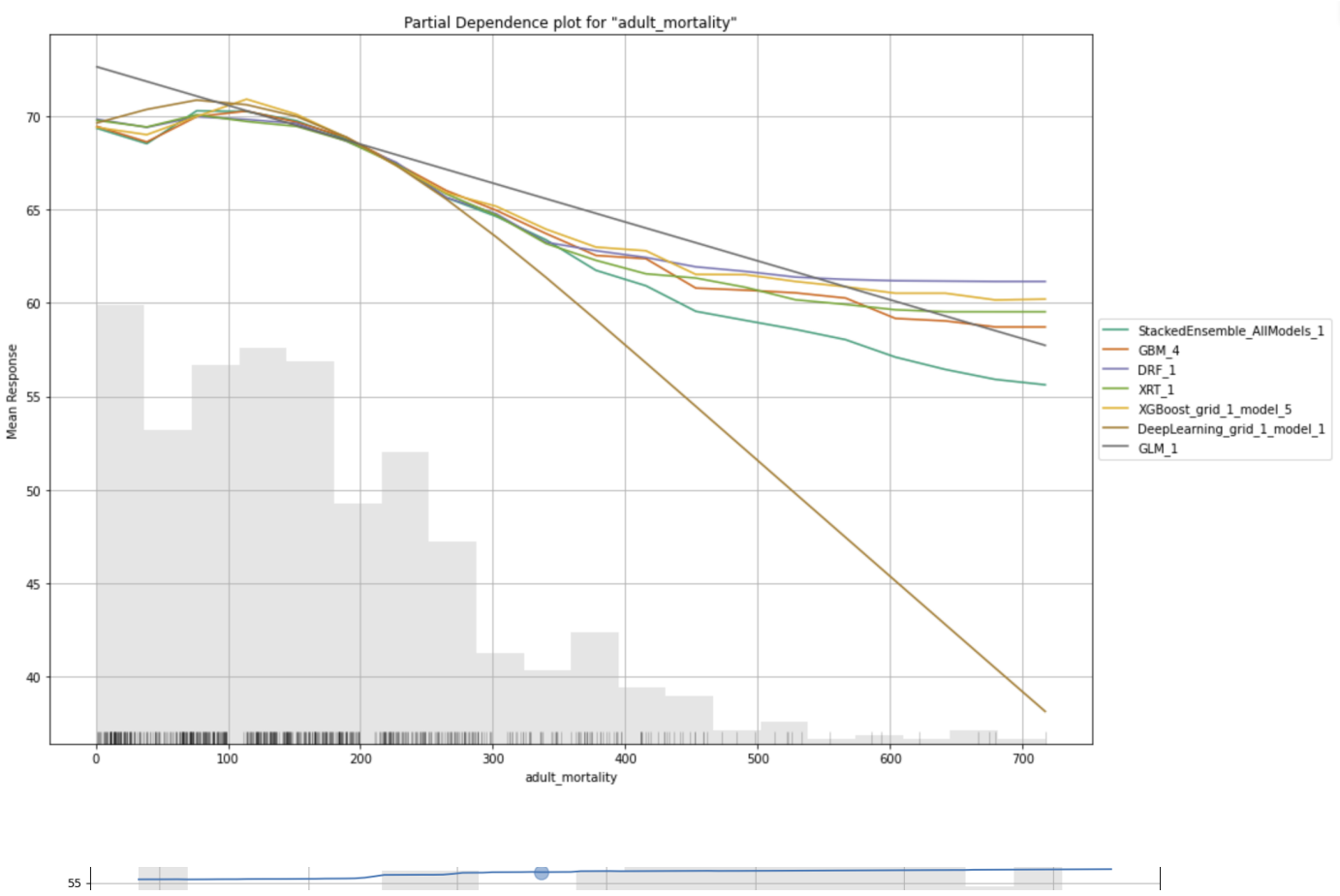


INFERENCE 1 : `hiv/aids`

> All models identified that when the levels of hiv/aids are high, the life expectancy tends to decrease.

> This indicates a **negative relationship** between `hiv/aids` and `life_expectancy`.

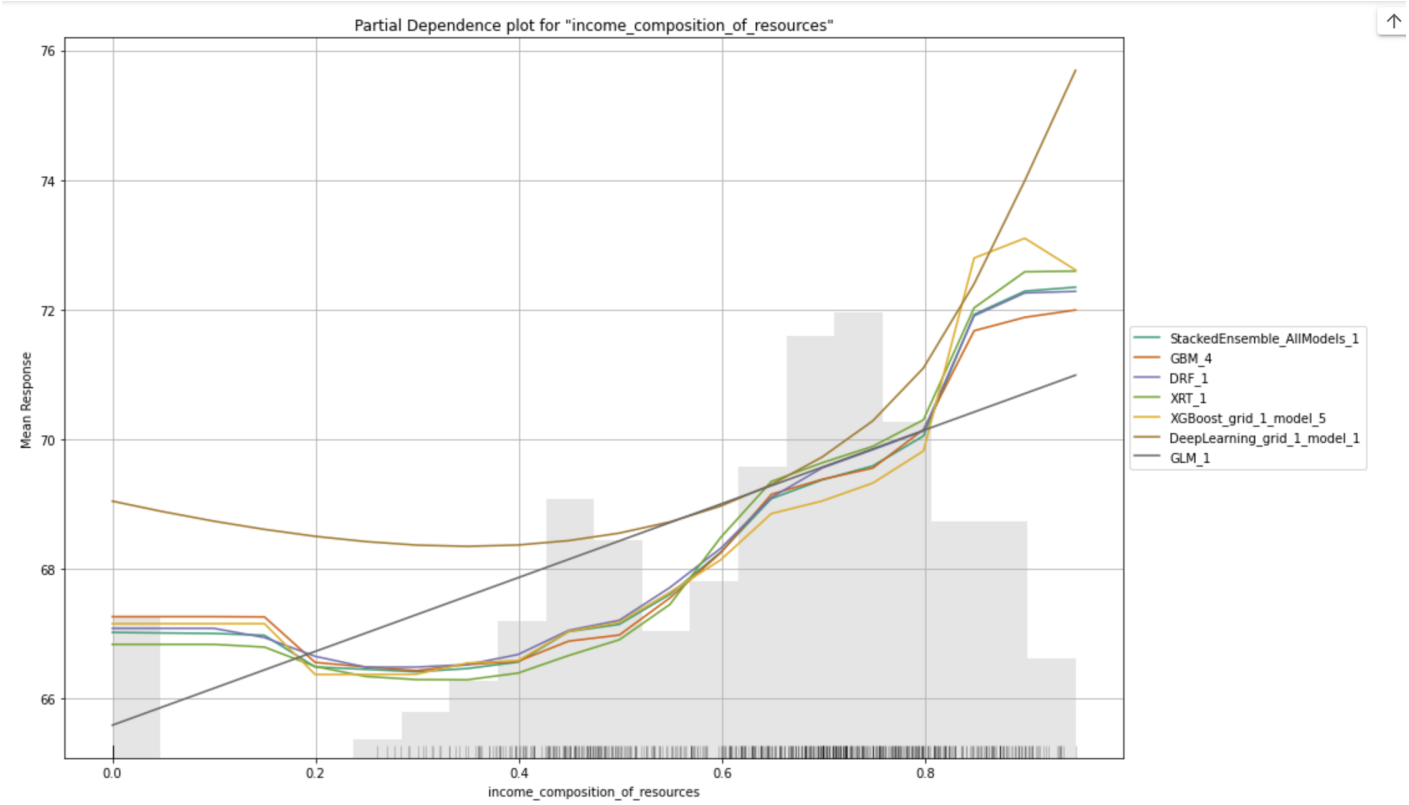
OBSERVATION 2 : `adult_mortality`



INFERENCE 2 : `adult_mortality`

> The models accurately identified the **inverse relationship** between `adult_mortality` and `life_expectancy`, meaning that when adult mortality rates are high, life expectancy is lower.

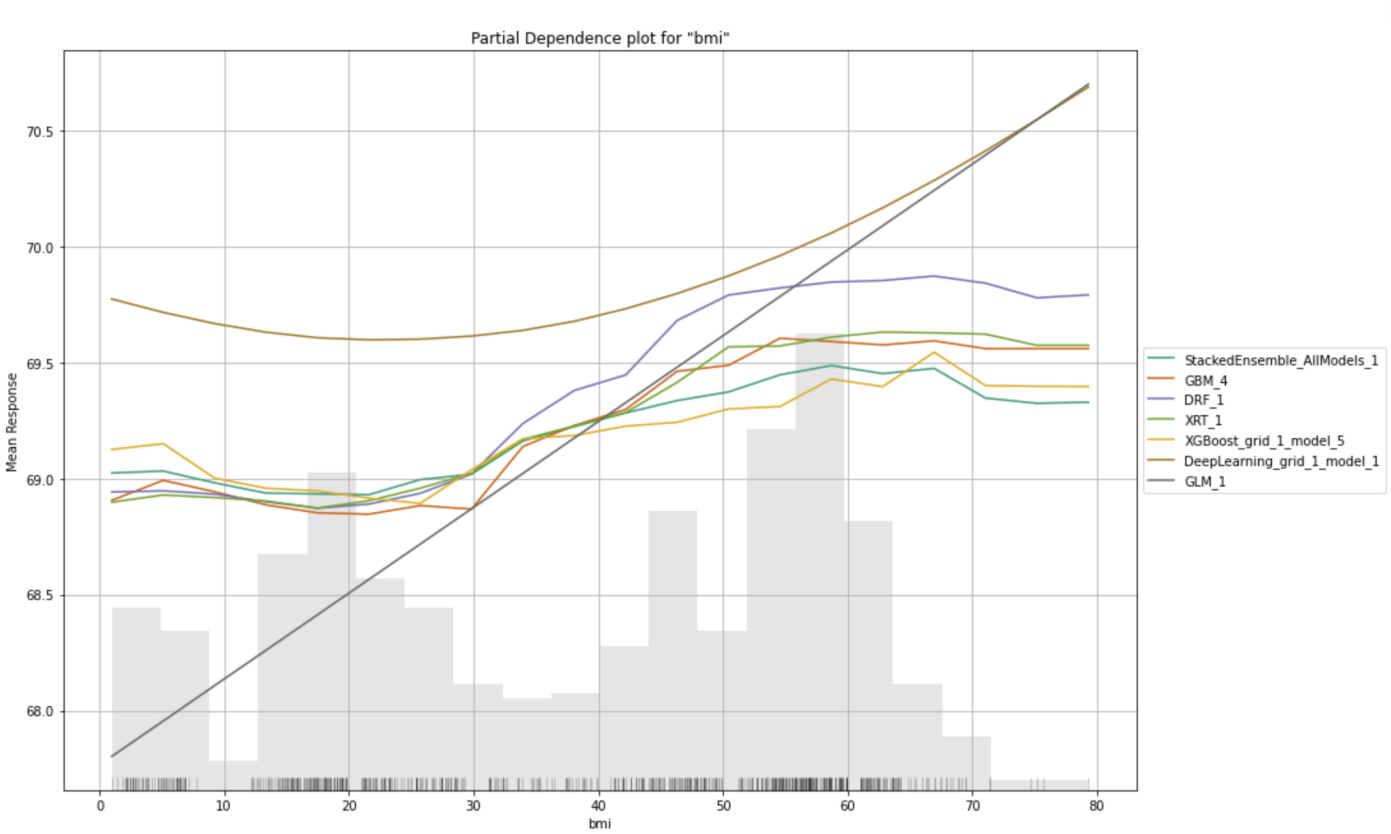
OBSERVATION 3 : `income_composition_of_resources`



INFERENCE 3: `income_composition_of_resources`

> All models accurately identified the **positive relationship** between `income_composition_of_resources` and `life_expectancy`, indicating that higher values of income_composition_of_resources lead to higher life_expectancy.

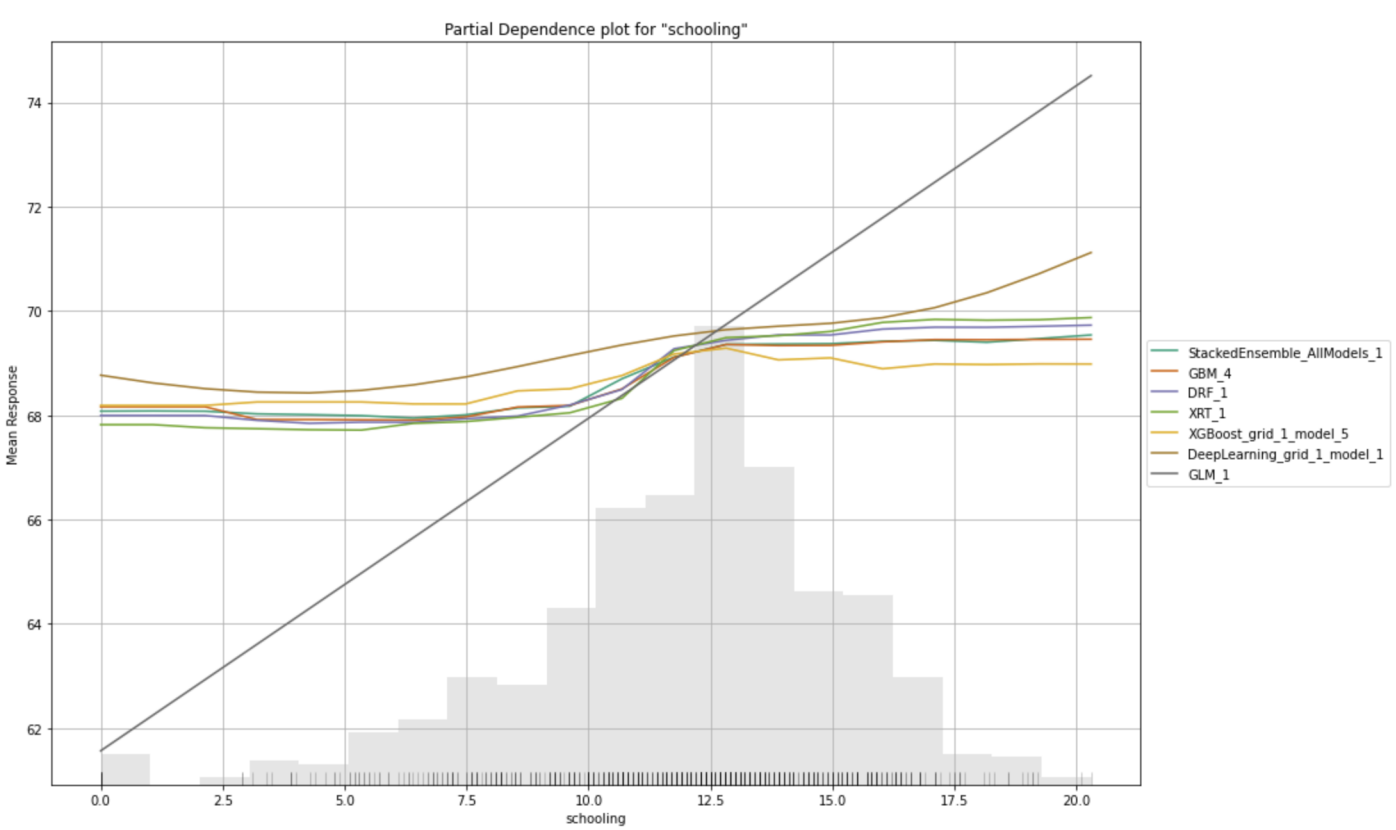
OBSERVATION 4 : `bmi`



INFERENCE 4: `bmi`

> All models accurately identified the **positive relationship** between `bmi` and `life_expectancy`, indicating that higher values of income_composition_of_resources lead to higher life_expectancy.

OBSERVATION 5 : `schooling`



INFERENCE 5: `schooling`

> All models accurately identified the **positive relationship** between `schooling` and `life_expectancy`, indicating that higher values of income_composition_of_resources lead to higher life_expectancy.

OVERALL CONCLUSION WITH H20 IMPLEMENTATION

---

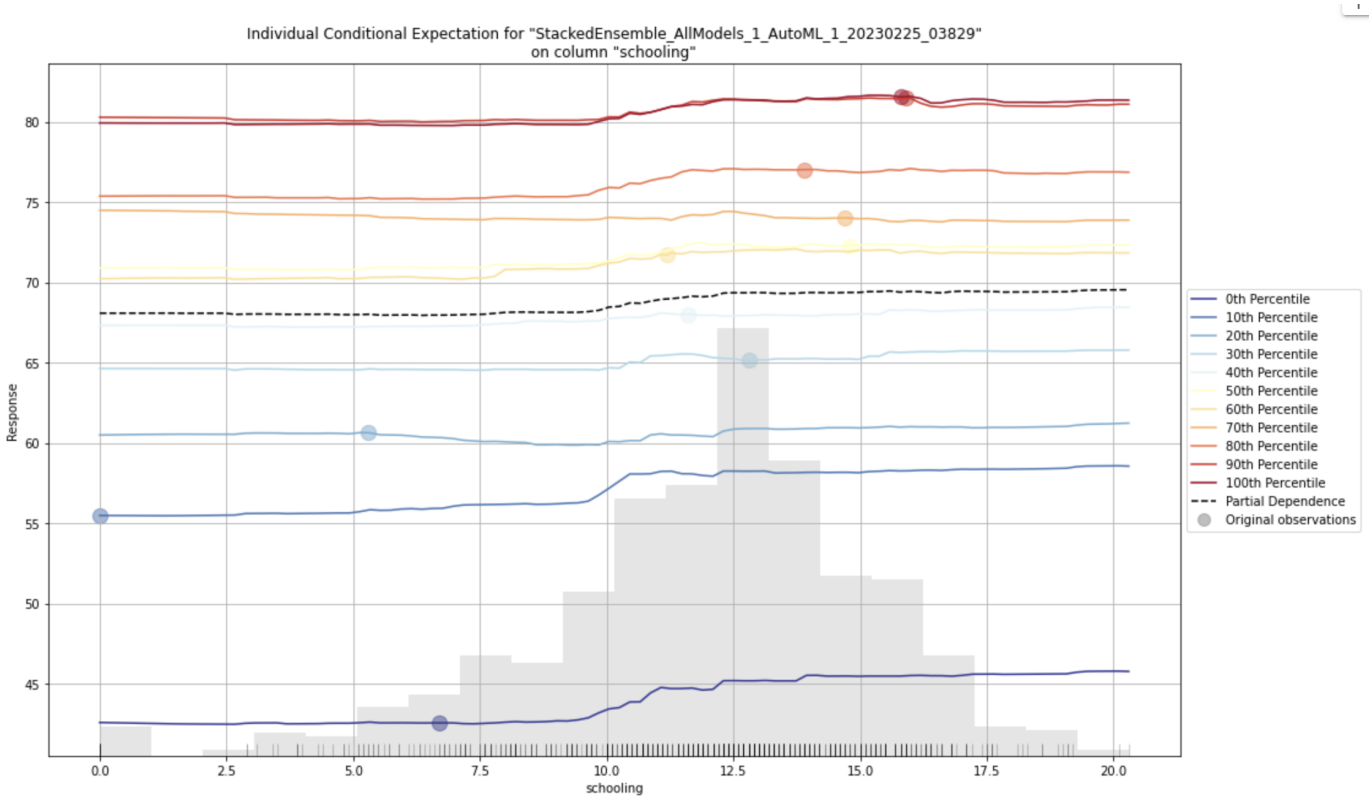
1. `hiv/aids`, `income_composition_of_resources`, `adult_mortality`, `schooling` and `thinness_5_to_9_years` are the most responsive features, according to the Variable Importance Plot.

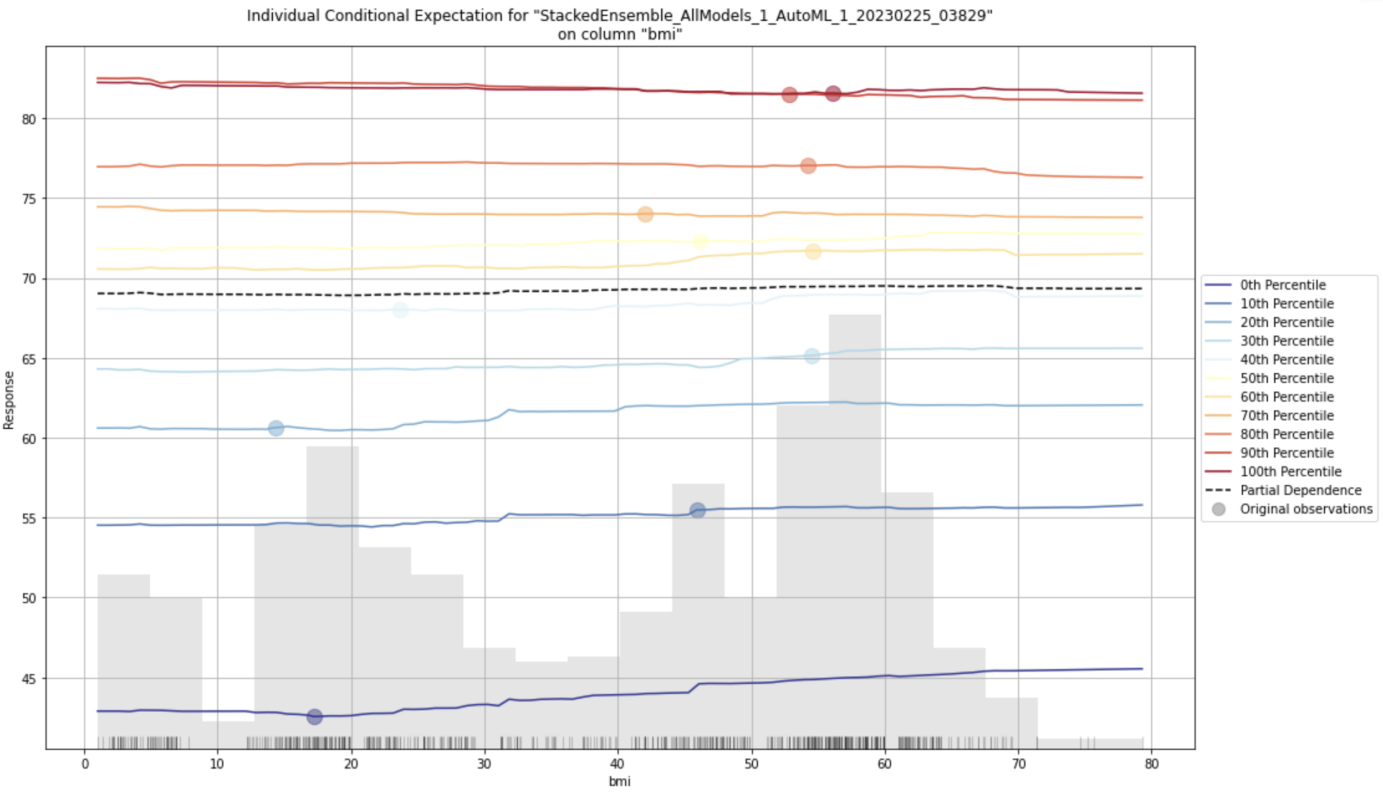
2. The **StackedEnsemble_BestOfFamily_2 model** performed the best out of all models.

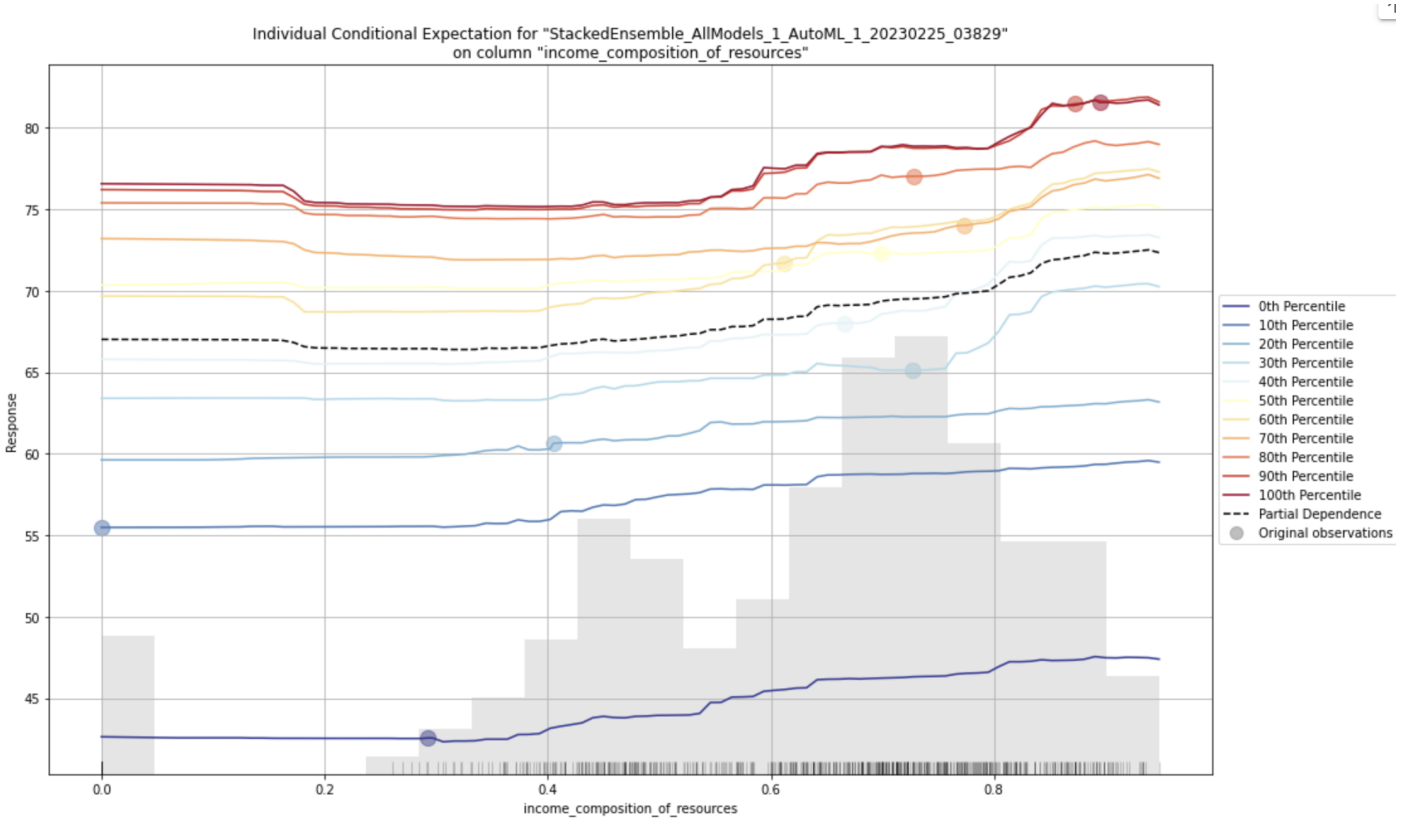
3. The **GLM_1** model did not perform as well as other models and had low predictive power.

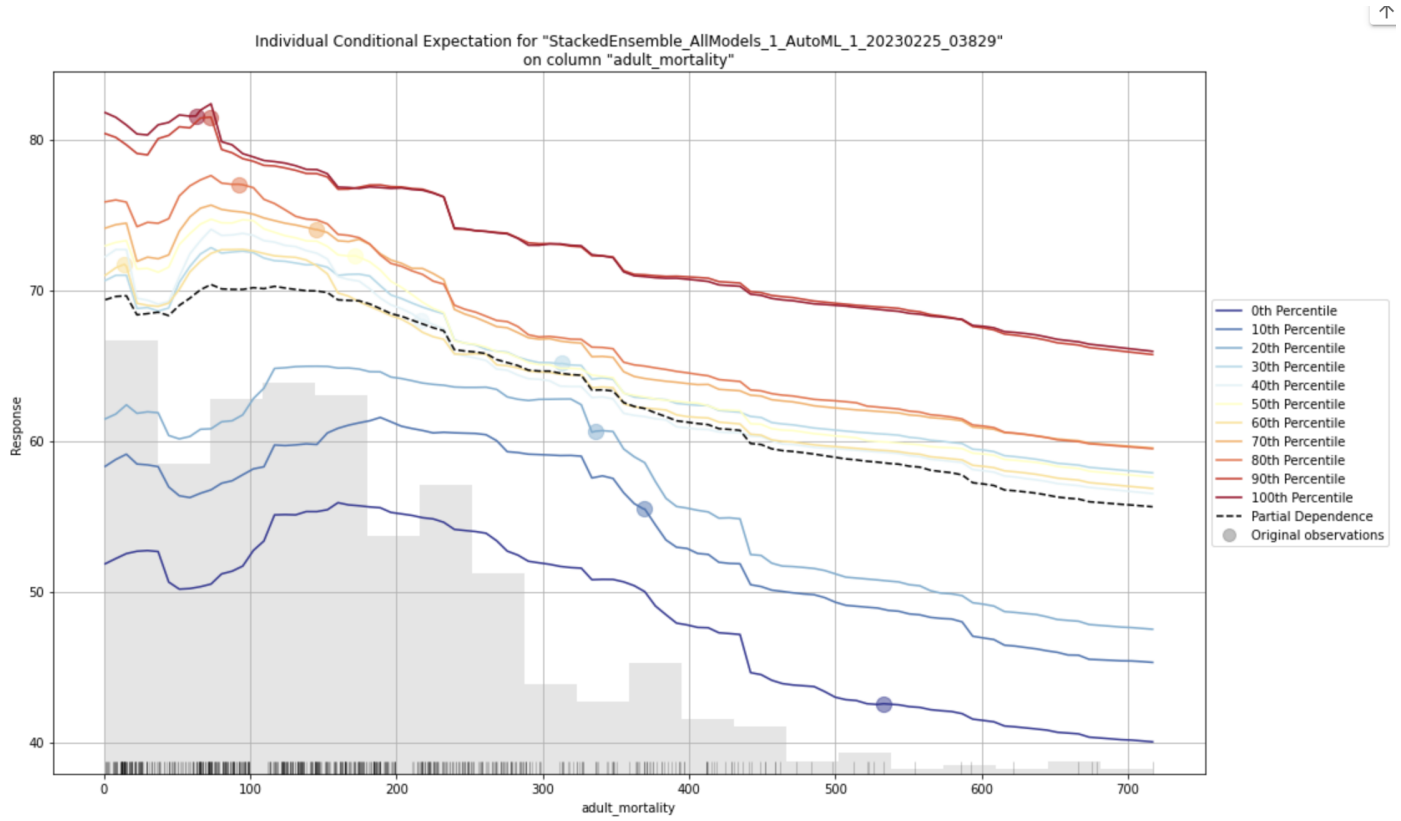
7. INDIVIDUAL CONDITIONAL EXCEPTION (ICE)

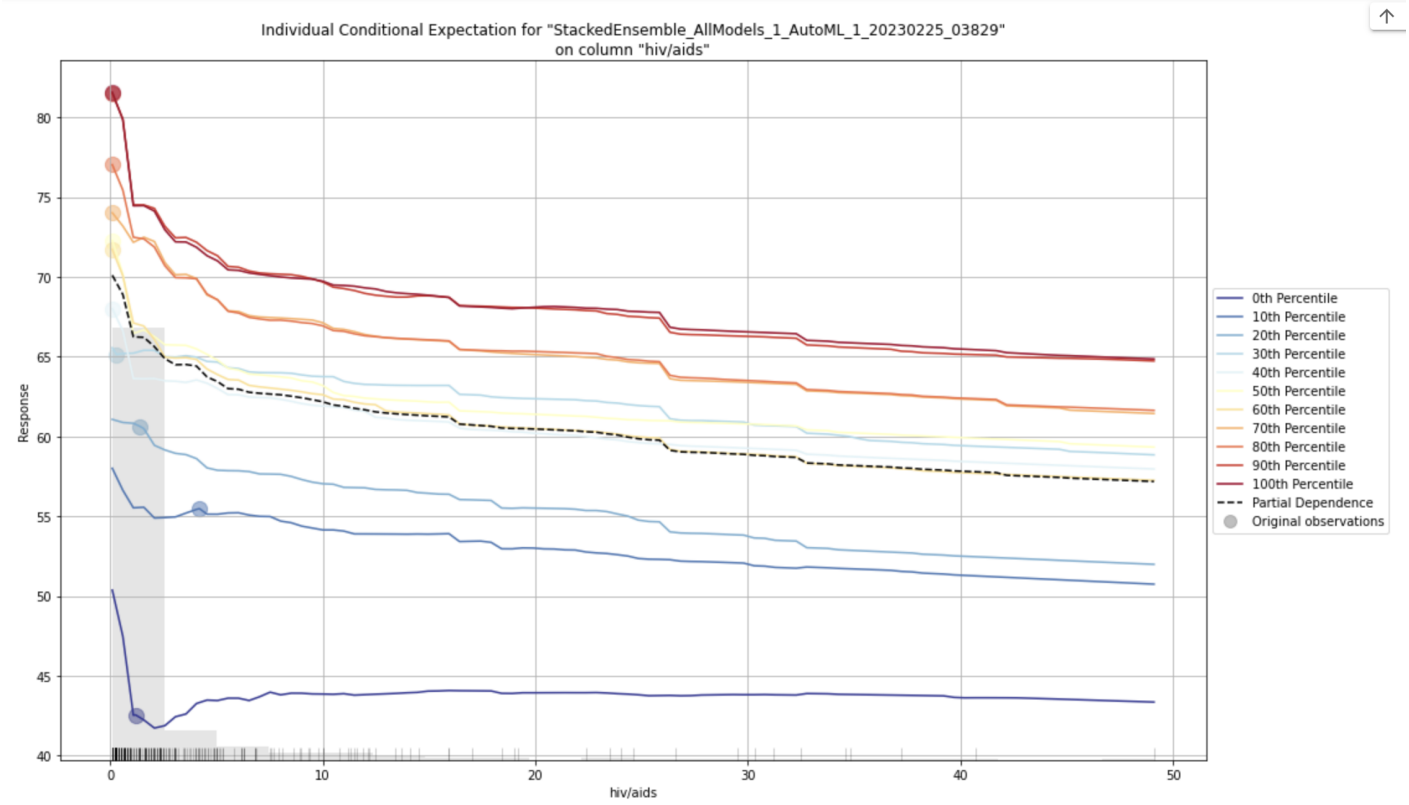
1. It is a technique to visualize the relationship between a single variable and the target variable, while holding other variables constant.
2. ICE plots help in understanding how the predictions change for different values of a single variable.
3. ICE Plot displays one line per instance that shows how the instance's prediction changes when a feature changes.


Individual Conditional Expectation for "StackedEnsemble_AllModels_1_AutoML_1_20230225_03829"
on column "schooling"
Response
schooling
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations


Individual Conditional Expectation for "StackedEnsemble_AllModels_1_AutoML_1_20230225_03829"
on column "bmi"
Response
bmi
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations


Individual Conditional Expectation for "StackedEnsemble_AllModels_1_AutoML_1_20230225_03829"
on column "income_composition_of_resources"
Response
income_composition_of_resources
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations


Individual Conditional Expectation for "StackedEnsemble_AllModels_1_AutoML_1_20230225_03829"
on column "adult_mortality"
Response
adult_mortality
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations



INFERENCE

> There is a consistent effect across all instances.

HYPER-PARAMETER TUNING

> 1. Hyperparameter tuning involves selecting the best set of hyperparameters for a machine learning algorithm.
> 2. Hyperparameters are settings or values that are set before training a model and can have a significant impact on its performance.
> 3. The goal of hyperparameter tuning is to find the set of hyperparameters that results in the best performance on a validation set. Grid search, random search, and Bayesian optimization are common techniques used for hyperparameter tuning.
> 4. Hyperparameter tuning can be time-consuming and computationally expensive, but it can lead to significant improvements in model performance.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X1 = df_model_original[columns]
y1 = df_model_original['winz_life_expectancy']


X_train, X_test, y_train, y_test= train_test_split(X1, y1, test_size = 0.3, random_st
 #split train and test sets
#train, test = h2o_df.split_frame(ratios=[.75])

#X1, y1 = boston.data, boston.target
dmatrix = xgb.DMatrix(data=X_test, label=y_test)
params={ 'objective':'reg:squarederror',
         'max_depth': 6,
         'colsample_bylevel':0.5,
         'learning_rate':0.01,
         'random_state':20}
cv_results = xgb.cv(dtrain=dmatrix, params=params, nfold=10, metrics={'rmse'}, as_pan
print('AFTER HYPER-PARAMETER TUNING RMSE: %.2f' % cv_results['test-rmse-mean'].min())
## Result : RMSE: 2.69
```

```
AFTER HYPER-PARAMETER TUNING RMSE: 0.20
```

MODEL INTERPRETABILITY

> Model interpretability in machine learning refers to the ability to explain how a machine learning model works and how it makes its predictions. It is an important aspect of machine learning, especially when the model is used in real-world applications where it is crucial to understand the factors that are driving the model's decisions.

> Model interpretability can help to increase trust in the model, as it provides insight into how the model is making predictions. This is particularly important in areas

> such as healthcare, where decisions based on machine learning models can have a significant impact on people's lives.

> There are various techniques and tools available for model interpretability, including:

> 1. **Feature importance:**This technique identifies the most important features that contribute to the model's predictions.
> 2. **Partial dependence plots:** This technique shows the relationship between a target variable and a feature while holding all other features constant.
> 3. **SHAP (SHapley Additive exPlanations):** This technique is a game-theoretic approach to explain the output of any machine learning model.
> 4. **LIME (Local Interpretable Model-agnostic Explanations):** This technique explains individual predictions by approximating the model locally with a simpler model.
> 5. **Decision trees:** These are a simple and interpretable model that can be used to explain how the model makes its predictions.

REFERENCES

---

https://www.who.int/data/gho/data/themes/mortality-and-global-health-estimates/ghe-life-expectancy-and-healthy-life-expectancy

https://towardsdatascience.com/model-interpretability-and-explainability-27fe31cc0688

```
df_life_expectancy = df_encoder_original_model
```

Defining Features of the Target Variables

```
x = ['year',
 'adult_mortality',
 'infant_deaths',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'measles',
 'under-five_deaths',
 'polio',
 'total_expenditure',
 'diphtheria',
```

```
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',
 'schooling',
 'Developed',
 'Developing']

y = ['life_expectancy']
```

Dataframe containing independant features

```
X = df_life_expectancy[x]
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 20 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   year                             2413 non-null   int64  
 1   adult_mortality                  2413 non-null   float64
 2   infant_deaths                    2413 non-null   int64  
 3   alcohol                          2413 non-null   float64
 4   percentage_expenditure           2413 non-null   float64
 5   hepatitis_b                      2413 non-null   float64
 6   measles                          2413 non-null   int64  
 7   under-five_deaths                2413 non-null   int64  
 8   polio                            2413 non-null   float64
 9   total_expenditure                2413 non-null   float64
 10  diphtheria                       2413 non-null   float64
 11  hiv/aids                         2413 non-null   float64
 12  gdp                              2413 non-null   float64
 13  population                       2413 non-null   float64
 14  thinness_1-19_years              2413 non-null   float64
 15  thinness_5-9_years               2413 non-null   float64
 16  income_composition_of_resources  2413 non-null   float64
 17  schooling                        2413 non-null   float64
 18  Developed                        2413 non-null   uint8  
 19  Developing                       2413 non-null   uint8  
dtypes: float64(14), int64(4), uint8(2)
memory usage: 362.9 KB
```

Dataframe containing dependant feature - life_expectancy

```
Y = df_life_expectancy[y]
Y.head()
```

	life_expectancy
16	77.8
32	75.6
48	52.4
64	76.4
80	76.3

NOTES

> random_state is a parameter in the train_test_split() function that allows you to specify a seed value for the random number generator used to split the data into training and testing sets.

> By setting a value for random_state, you can ensure that the same set of random indices is generated each time you run the code. This is useful for reproducibility, as it ensures that the same split is obtained each time the code is run, making it easier to compare the results of different models.

> For example, in the code I provided, random_state=32 is used to set the seed value for the random number generator. This means that every time the code is run with random_state=32, the same set of random indices will be generated, resulting in the same training and testing sets.

```
print(f"Mean value of Life_Expectancy after imputing with median values (in years): {
```

```
Mean value of Life_Expectancy after imputing with median values (in years): life
dtype: float64
```

Fit a linear model and interpret the regression coefficients

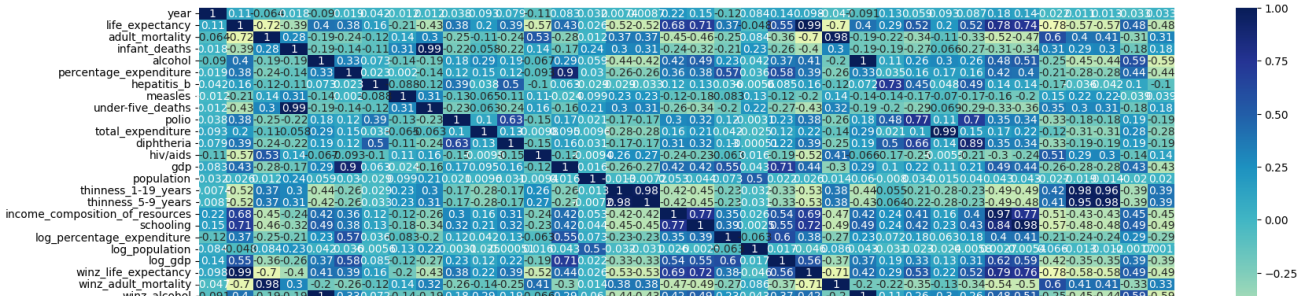
Linear Regression Model

> OLS is a statistical technique that helps to find the best-fitting line through a set of data points by minimizing the differences between the actual values and the predicted values of a dependent variable based on the independent variables. This method is widely used in economics, finance, social sciences, and other fields to estimate the relationships between variables and to make predictions based on those relationships.

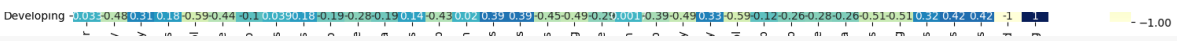
Visualizing the Correlation

```
#the heat map of the correlation
plt.figure(figsize=(20,7))
sns.heatmap(df_life_expectancy.corr(), annot=True, cmap='YlGnBu')
```

```
The default value of numeric_only in DataFrame.corr is deprecated. In a future v
<Axes: >
```



Splitting the data into Train, Test and Split




```
X1 = sm.add_constant(X)  # add constant term to X

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, Y, test_size=0.2, random_stat
```

Fitting the OLS Model

```
# Fit OLS model on training data
model = sm.OLS(y_train, X_train).fit()
```

```
reg_summary = model.summary()
print(reg_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        life_expectancy   R-squared:                       0.816
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     446.2
Date:                Mon, 24 Apr 2023   Prob (F-statistic):               0.00
Time:                        13:36:45   Log-Likelihood:                -5350.0
No. Observations:                1930   AIC:                         1.074e+04
Df Residuals:                    1910   BIC:                         1.085e+04
Df Model:                          19
Covariance Type:            nonrobust
==============================================================================
                                 coef    std err          t      P>|t|
------------------------------------------------------------------------------
const                         94.6068     28.166      3.359      0.001
```

```
year                             -0.0412      0.021     -1.950      0.051
adult_mortality                  -0.0197      0.001    -19.299      0.000
infant_deaths                     0.1549      0.030      5.225      0.000
alcohol                           0.0352      0.031      1.128      0.259
percentage_expenditure        -3.895e-05      0.000     -0.359      0.720
hepatitis_b                      -0.0150      0.005     -3.210      0.001
measles                          -0.0007      0.001     -1.361      0.174
under-five_deaths                -0.1327      0.020     -6.605      0.000
polio                             0.0270      0.006      4.829      0.000
total_expenditure                 0.0882      0.040      2.195      0.028
diphtheria                        0.0294      0.006      4.981      0.000
hiv/aids                         -0.4398      0.021    -20.586      0.000
gdp                           5.424e-05   1.66e-05      3.261      0.001    2
population                     2.69e-09      6e-09      0.448      0.654   -9
thinness_1-19_years              -0.1598      0.112     -1.425      0.154
thinness_5-9_years               -0.0544      0.110     -0.494      0.621
income_composition_of_resources   6.4274      0.704      9.132      0.000
schooling                         0.6414      0.048     13.328      0.000
Developed                        47.9678     14.074      3.408      0.001
Developing                       46.6391     14.094      3.309      0.001
==============================================================================
Omnibus:                      159.677   Durbin-Watson:                   2.028
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              899.052
Skew:                          -0.109   Prob(JB):                   5.94e-196
Kurtosis:                       6.336   Cond. No.                    1.65e+22
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
[2] The smallest eigenvalue is 2.06e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

NOTES:

> R-squared (R^2) is a statistical measure that represents the proportion of the variance in the dependent variable (i.e., the target variable) that is explained by the independent variables (i.e., the predictor variables) in a regression model. In other words, it is a measure of how well the model fits the data.

> R-squared values range from 0 to 1, with higher values indicating better fit. An R-squared value of 1 indicates that all of the variance in the dependent variable is explained by the independent variables and that the model fits the data perfectly.

Displaying Regression Coefficients

> Regression coefficients are values that measure the association between an independent variable and a dependent variable in a linear regression model. Specifically, regression coefficients represent the slope of the line in the linear regression equation that describes the relationship between the dependent variable and the independent variable.

> In a multiple linear regression model with more than one independent variable, there are multiple regression coefficients, each of which measures the effect of one independent variable on the dependent variable, while holding all other independent variables constant.

> Each coefficient represents the change in the dependent variable for each one-unit change in the corresponding independent variable, while all other independent variables are held constant.

```
# Print regression coefficients
print(model.params)
```

```
const                              9.460683e+01
year                              -4.119763e-02
adult_mortality                   -1.973662e-02
infant_deaths                      1.549430e-01
alcohol                            3.523816e-02
percentage_expenditure            -3.895228e-05
hepatitis_b                       -1.504937e-02
measles                           -7.181804e-04
under-five_deaths                 -1.327418e-01
polio                              2.701775e-02
total_expenditure                  8.824110e-02
diphtheria                         2.941232e-02
hiv/aids                          -4.397686e-01
gdp                                5.423852e-05
population                         2.690218e-09
thinness_1-19_years               -1.598013e-01
thinness_5-9_years                -5.442914e-02
income_composition_of_resources    6.427401e+00
schooling                          6.413780e-01
Developed                          4.796778e+01
Developing                         4.663905e+01
dtype: float64
```

Visualizing Regression Coefficients

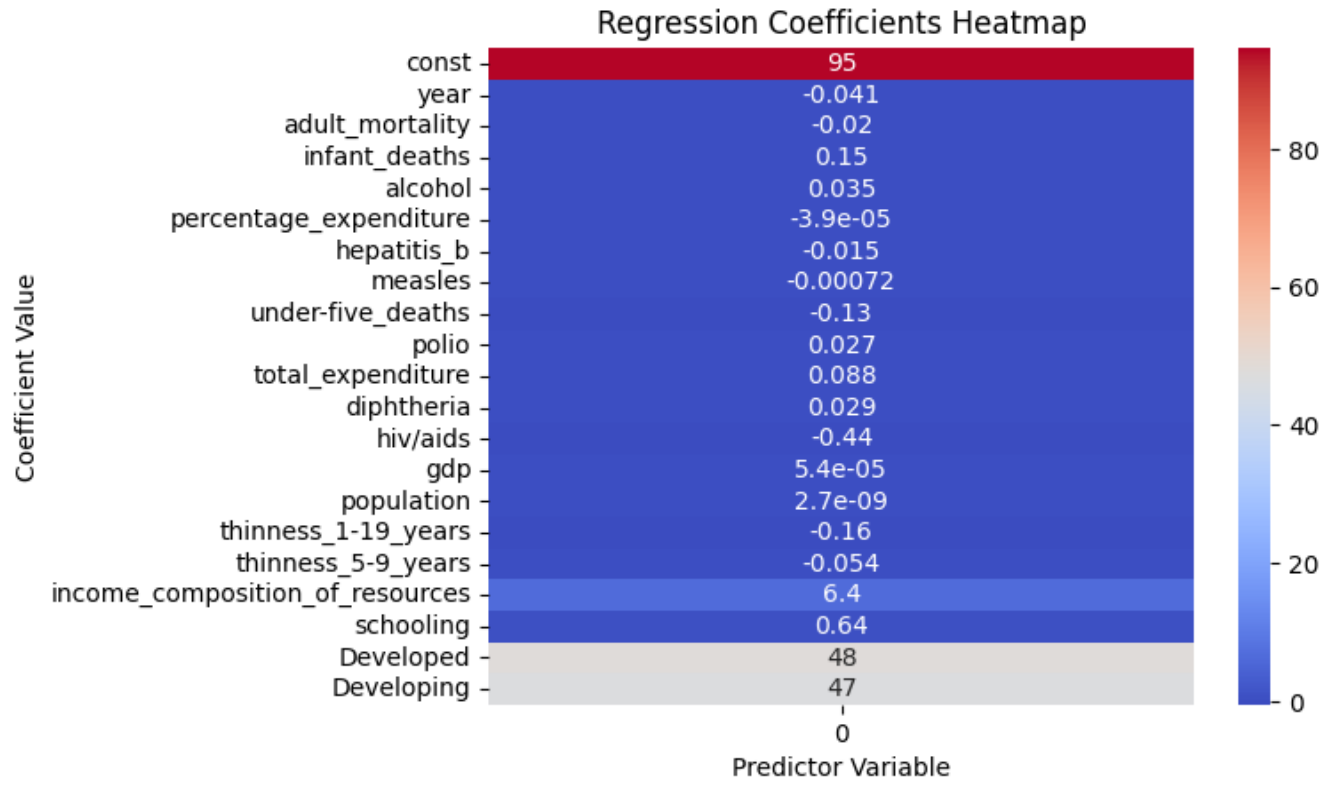
```
#Storing the coefficients obtained from linear regression
coeff = model.params
df_coeff = coeff.to_frame()
df_coeff
```

	0
const	9.460683e+01
year	-4.119763e-02
adult_mortality	-1.973662e-02
infant_deaths	1.549430e-01
alcohol	3.523816e-02
percentage_expenditure	-3.895228e-05
hepatitis_b	-1.504937e-02
measles	-7.181804e-04
under-five_deaths	-1.327418e-01
polio	2.701775e-02
total_expenditure	8.824110e-02
diphtheria	2.941232e-02
hiv/aids	-4.397686e-01
gdp	5.423852e-05
population	2.690218e-09
thinness_1-19_years	-1.598013e-01
thinness_5-9_years	-5.442914e-02
income_composition_of_resources	6.427401e+00
schooling	6.413780e-01
Developed	4.796778e+01
Developing	4.663905e+01

NOTES:

to_frame() converts a Pandas Series of regression coefficients returned by the params attribute of a statsmodels OLS model to a DataFrame.

```
# Create a heatmap of the coefficients
sns.heatmap(df_coeff, cmap='coolwarm', annot=True, cbar=True)

# Set the plot title and axes labels
plt.title('Regression Coefficients Heatmap')
plt.xlabel('Predictor Variable')
plt.ylabel('Coefficient Value')

# Show the plot
plt.show()
```



INFERENCE

These estimated coefficients can be interpreted as the amount by which the dependent variable is expected to change for a one-unit increase in the corresponding independent variable, holding all other variables constant. Additionally, the sign of the coefficient (positive or negative) indicates the direction of the relationship between the independent variable and the dependent variable.

---

1. `adult_mortality` The value of -1.988565e-02 for the regression coefficient of adult_mortality in a linear regression model suggests that there is a negative relationship between adult_mortality and the `life_expectancy`.
2. `alcohol` The value of 5.116640e-02 for the regression coefficient of alcohol in a linear regression model suggests that there is a positive relationship between alcohol and the `life_expectancy`
3. `under-five_deaths` The value of -6.907899e-02 for the regression coefficient of under-five_deaths in a linear regression model suggests that there is a negative relationship between under-five_deaths and the `life_expectancy`
4. `schooling` The value of 6.923112e-01 for the regression coefficient of schooling in a linear regression model suggests that there is a positive relationship between schooling and the `life_expectancy`

MODEL PREDICTION

```
prediction(X_train, X_test, y_train, y_test)
```

```
Training MAE:  2.8319461978622695
Testing MAE:  2.9733567049538507

Training RMSE:  3.8693137416946244
Testing RMSE:  3.908834522345046
```

INFERENCE

MEAN ABSOLUTE ERROR (MAE)

> The mean absolute error (MAE) is a metric used to evaluate the performance of a regression model. It represents the average absolute difference between the predicted values and the actual values in the data set. The lower the MAE, the better the performance of the model.

> the training MAE is 3.007, which means that on average, the model's predictions on the training data are off by about 3 units. The testing MAE is 3.014, which means that on average, the model's predictions on the testing data are off by about 3 units. The fact that the testing MAE is similar to the training MAE suggests that the model is not overfitting to the training data, and is generalizing well to new, unseen data.

ROOT MEAN SQUARED ERROR (RMSE)

> The root mean squared error (RMSE) is a metric used to evaluate the performance of a regression model. It represents the square root of the average squared difference between the predicted values and the actual values in the data set. The lower the RMSE, the better the performance of the model.

> In this case, the training RMSE is 4.033, which means that on average, the model's predictions on the training data are off by about 4 units. The testing RMSE is 4.061, which means that on average, the model's predictions on the testing data are off by about 4 units. The fact that the testing RMSE is similar to the training RMSE suggests that the model is not overfitting to the training data, and is generalizing well to new, unseen data.

SHAP ANALYSIS - LINEAR REGRESSION

> SHAP (SHapley Additive exPlanations) is a method for explaining the output of any machine learning model by calculating the contribution of each feature to the final prediction. It is based on the concept of Shapley values from cooperative game theory and uses the idea of local explanations, which means explaining the prediction for a specific instance rather than the model as a whole.

> SHAP analysis provides feature importance values that show how much each feature contributes to the predicted outcome for a particular data point. It can also help identify interactions between features and reveal how changes in one feature impact the prediction.

> SHAP analysis can be applied to a variety of models, including linear regression, decision trees, random forests, and neural networks. It is a powerful tool for

> understanding how machine learning models make predictions and can be used to improve model interpretability, identify bias, and debug models.

REFERENCES

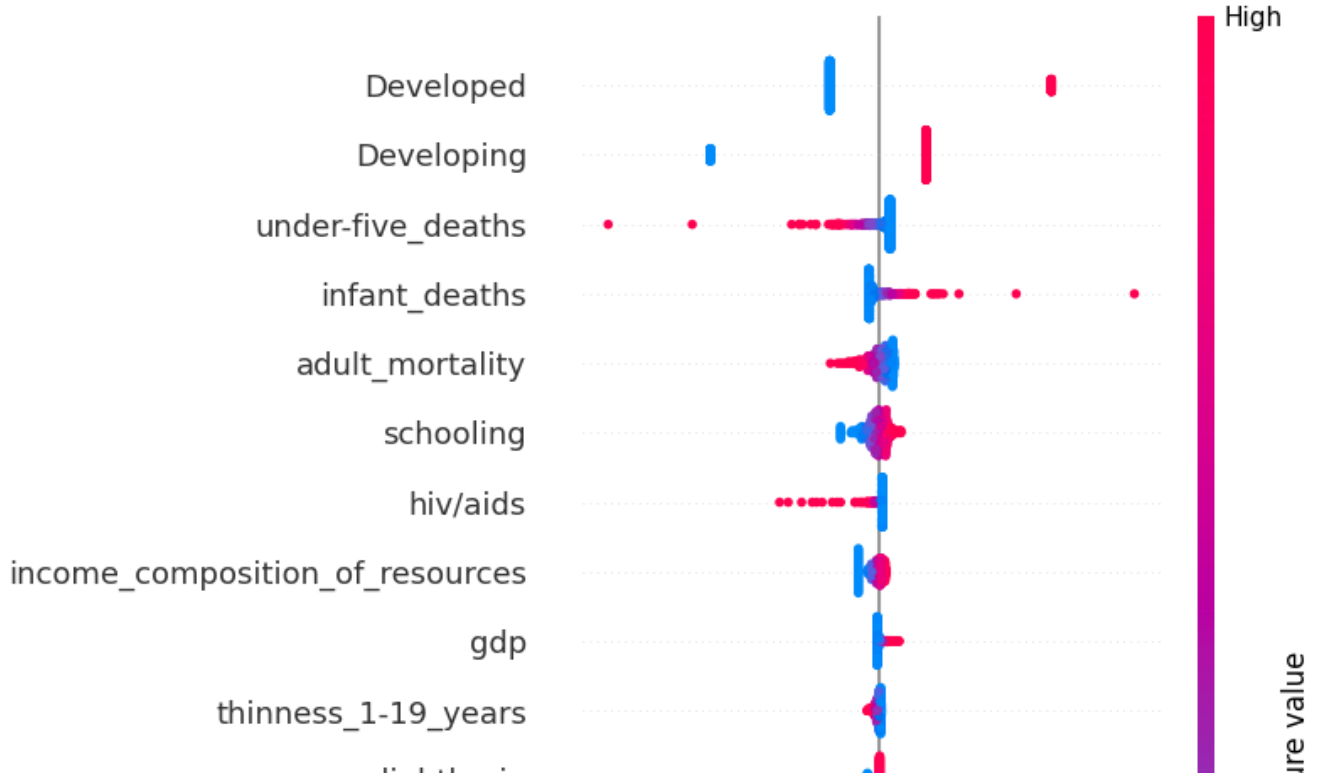
> https://towardsdatascience.com/introduction-to-shap-with-python-d27edc23c454

```
explainer = shap.Explainer(model.predict, X_train)
shap_values = explainer(X_test)
```

```
Permutation explainer: 484it [00:19, 13.60it/s]
```

```
#Summary plot
shap.summary_plot(shap_values, X_test)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be igr
```

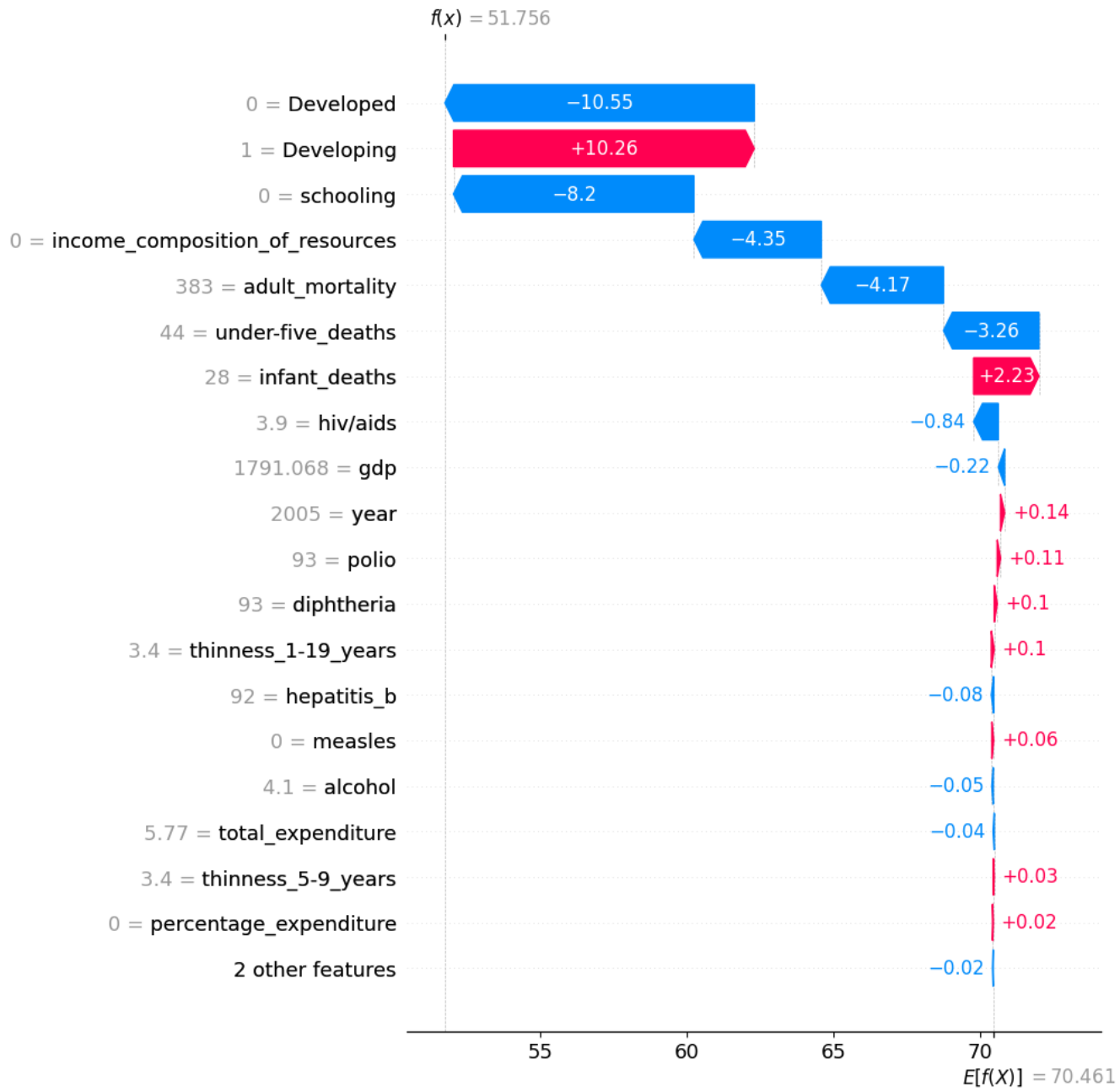


EXPLANATION : SHAP SUMMARY

1. High values of `Hiv/aids` and `under-five_deaths` have a **negative** impact on the outcome.

2. Other features such as `thinness`, `BMI`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.


```
#Waterfall Plot
shap.plots.waterfall(shap_values[0], max_display=20, show=True)
```



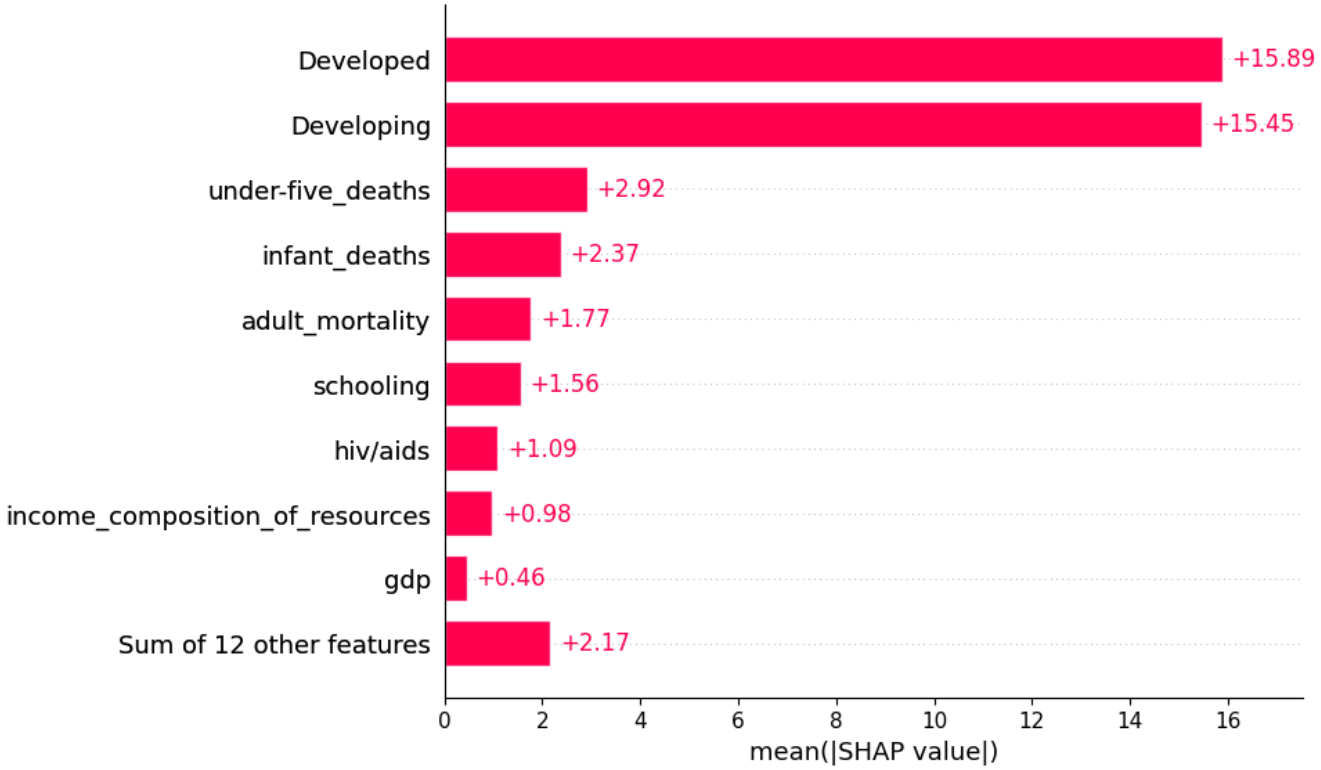
EXPLANATION : WATERFALL PLOT

> E[f(x)] = 69.39 gives the average predicted life_expectancy. f(x) = 62.586 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

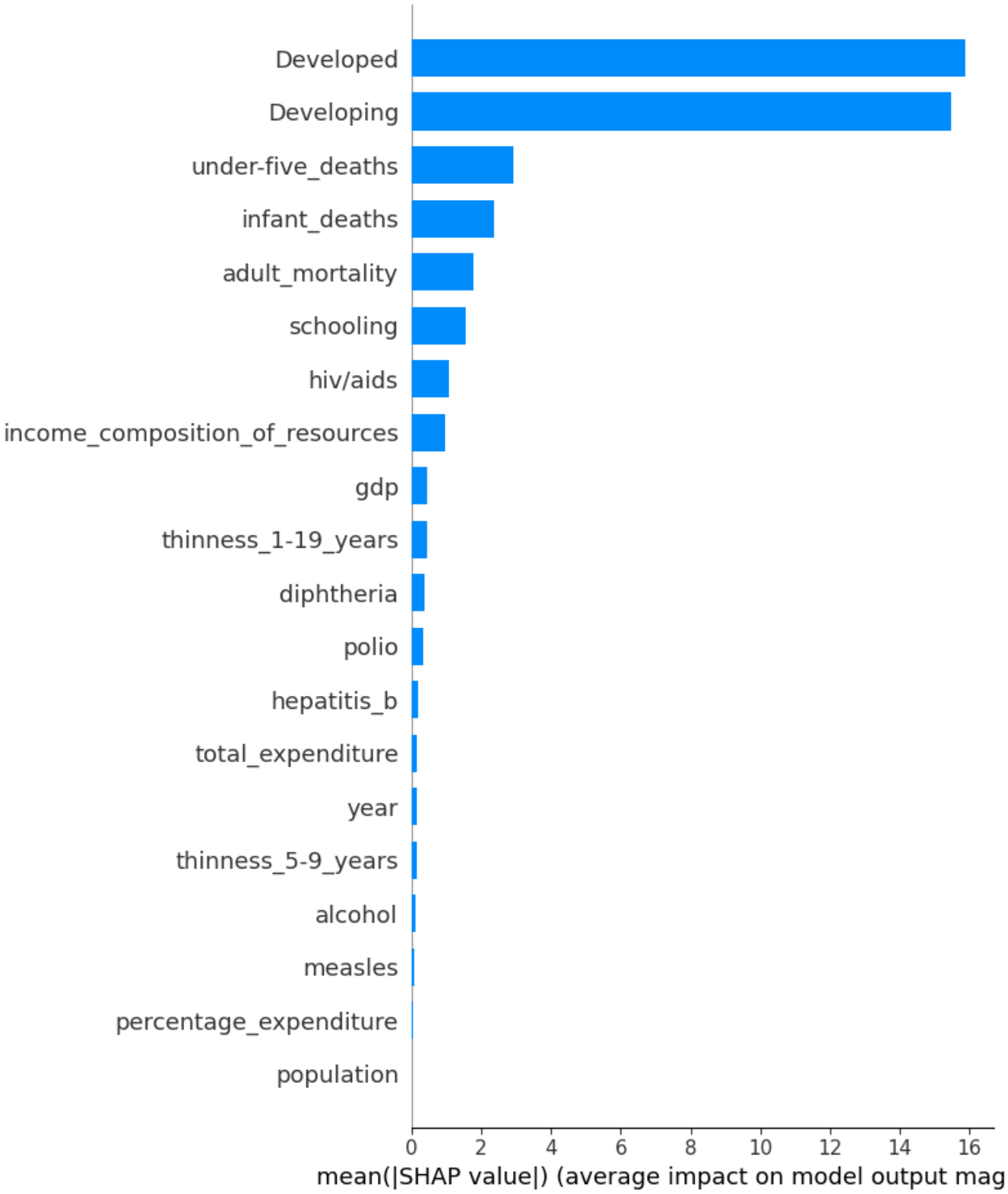
> SHAP values tell us how the features have contributed to the prediction when compared to the mean prediction. Large positive/negative values indicate that the feature had a significant impact on the model's prediction.

```
shap.initjs()
```

```
#Bar Plot
shap.plots.bar(shap_values)
```




```
#Bar Summary Plot
shap.summary_plot(shap_values, X_test, plot_type="bar")
```


Developed
Developing
under-five_deaths
infant_deaths
adult_mortality
schooling
hiv/aids
income_composition_of_resources
gdp
thinness_1-19_years
diphtheria
polio
hepatitis_b
total_expenditure
year
thinness_5-9_years
alcohol
measles
percentage_expenditure
population
0
2
4
6
8
10
12
14
16
mean(|SHAP value|) (average impact on model output mag

EXPLANATION : BAR PLOT VS SUMMARY PLOT

> The **shap.plots.bar()**function generates a bar plot to display the importance of each feature in the dataset according to their Shapley values.

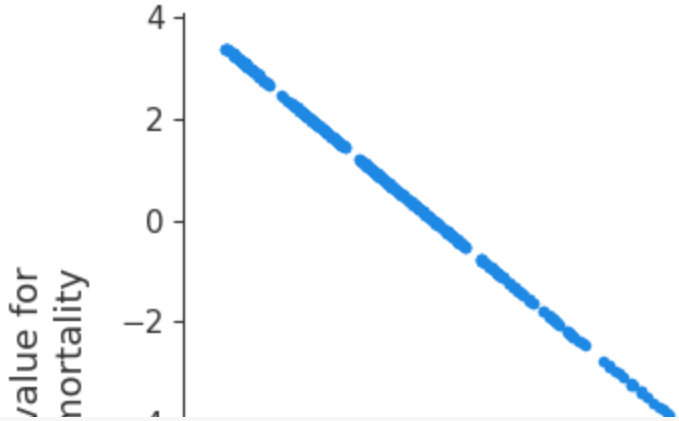
> **shap.plots.bar(shap_values)** generates a bar plot that shows the mean absolute SHAP value for each feature, averaged over all instances in the dataset. This plot provides a quick overview of the most important features and their direction of effect on the target variable.

> On the other hand, **shap.summary_plot(shap_values, X_test, plot_type="bar")** generates a more detailed bar plot that shows the distribution of SHAP values for each feature, along with their interaction effects. This plot provides a more granular view of the feature importance, as well as any potential interactions between features.

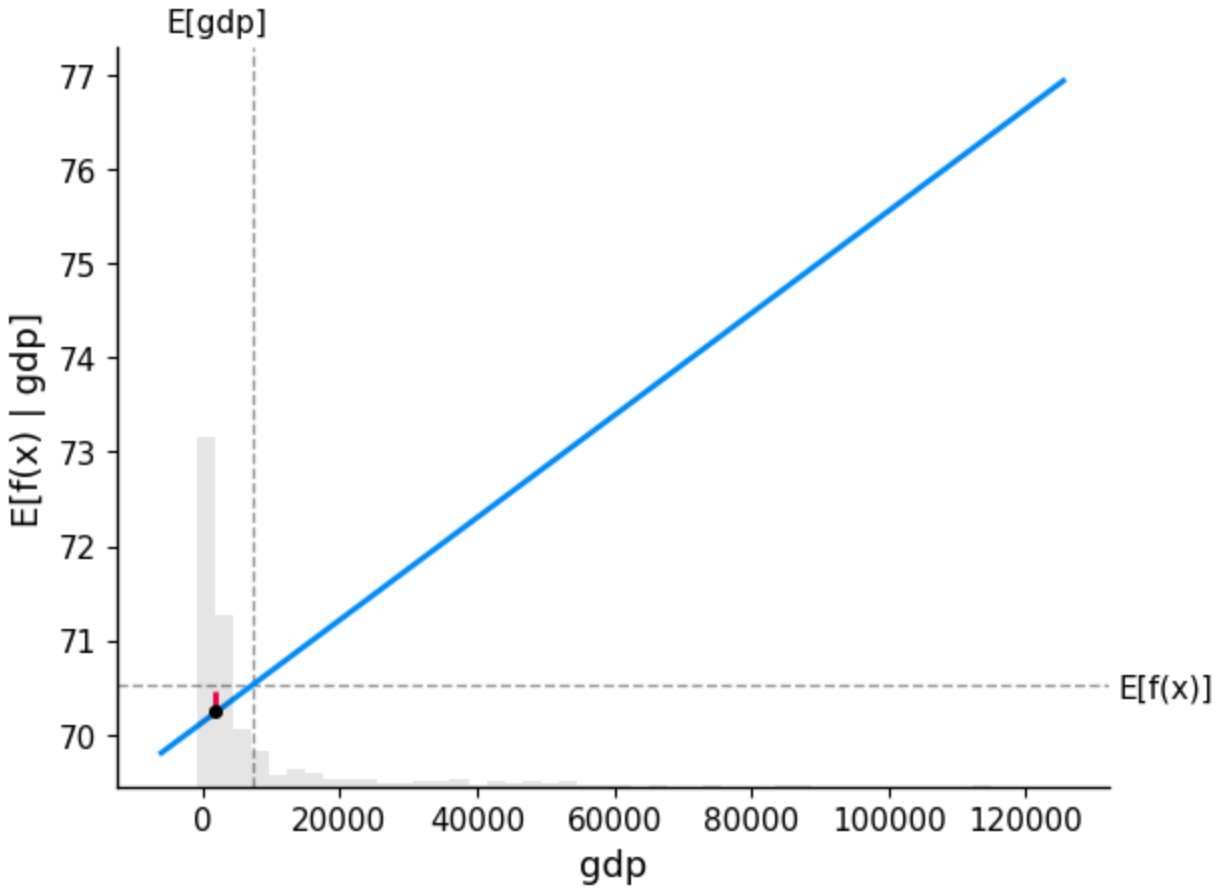
> In summary, shap.plots.bar(shap_values) is a quick way to visualize the most important features, while shap.summary_plot(shap_values, X_test, plot_type="bar") provides a more detailed view of the feature importance and interaction effects.

> In both of them `developed`, `developing`, `under-five-deaths` are significantly important features.

```
#Plot 1: adult_mortality

shap.plots.scatter(shap_values[:,"adult_mortality"])
```




```
partial_dependence_plot(model,'gdp', 0)
```



EXPLANATION : PARTIAL DEPENDANCY PLOT

---

1. The dependency plot tells us that the relationship is not perfectly linear. There is a significant decrease in the SHAP value of Adult mortality as seen the graph.

2. gdp is seen to increase linearly as SHAP value increases.

Fit a tree-based model and interpret the nodes

DECISION TREE

> **DecisionTreeRegressor** is a class in the scikit-learn library for performing regression using decision trees. It is a supervised machine learning algorithm that predicts the value of a continuous target variable by learning simple decision rules inferred from the features of the data. The decision tree splits the data into subsets based on the values of one of the features, and recursively partitions the subsets until it arrives at terminal nodes where it makes the final prediction.

Splitting the data into Train, Test and Split

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, Y, test_size=0.2, random_stat
```

Fitting the Decision Tree Model

```
tree_model = DecisionTreeRegressor(max_depth=3, random_state=42)
tree_model.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=3, random_state=42)
```

Model Prediction

```
prediction(X_train, X_test, y_train, y_test)
```

```
Training MAE:  2.8319461978622695
Testing MAE:  2.9733567049538507
```

```
Training RMSE:  3.8693137416946244
Testing RMSE:  3.908834522345046
```

INFERENCE

MEAN ABSOLUTE ERROR (MAE)

The training MAE is 2.9235 and the testing MAE is 3.0115. This means that, on average, the model's predictions are off by about 2.92 years and 3.01 years for the training and testing sets, respectively. The testing MAE is slightly higher than the training MAE, which indicates that the model may be slightly overfitting to the training data.

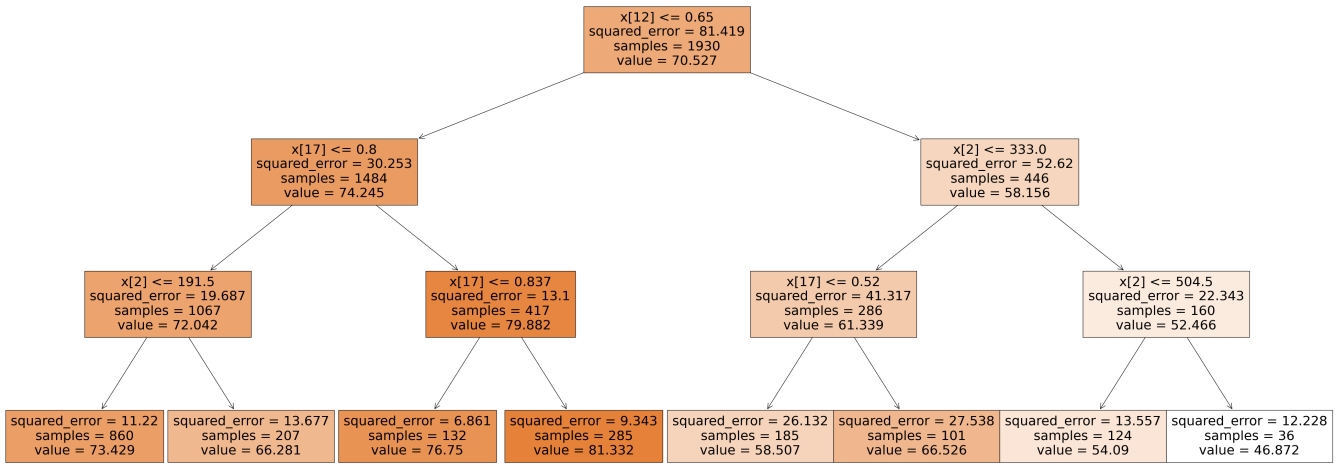
ROOT MEAN SQUARED ERROR (RMSE)

The training RMSE is 3.9129 and the testing RMSE is 4.1337. RMSE is a measure of the average magnitude of the errors in the predictions made by the model, with lower values indicating better performance. The training RMSE is lower than the testing RMSE, which suggests that the model may be overfitting to the training data. A higher testing RMSE indicates that the model's predictions are less accurate on unseen data.

Overall, the model appears to be performing reasonably well but may benefit from additional tuning to improve its performance on the testing set.

VISUALIZING THE DECISION TREE

```
# plot the decision tree
fig, ax = plt.subplots(figsize=(50, 20))
plot_tree(tree_model, filled=True, ax=ax)
plt.show()
```



NOTES:

---

1. Decision trees can be interpreted by examining the splits made at each node and the corresponding feature importance. The splits in a decision tree are based on the feature that provides the most information gain or reduction in impurity in the data. The impurity of a node is measured using a criterion such as Gini impurity or entropy.
2. To interpret a decision tree, we can follow these steps:
3. Start at the root node and examine the feature that was used to make the split. This feature will be the most important feature in the decision tree. Follow the path of the decision tree based on the splits until you reach a leaf node. The leaf node will give the final decision or classification based on the feature values of the input data.
4. Examine the feature importance of each node to understand which features are the most important in making decisions in the decision tree. The feature importance is calculated based on the number of times a feature is used to make a split in the decision tree.

RANDOM FOREST

Splitting the data into Train, Test and Split

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, Y, test_size=0.2, random_stat
```

Fitting the Model

```
# Fit the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=32)
rf.fit(X_train, y_train)
```

```
A column-vector y was passed when a 1d array was expected. Please change the sha
```

```
RandomForestRegressor(random_state=32)
```

Model Prediction

```
prediction(X_train, X_test, y_train, y_test)
```

```
Training MAE:  2.8319461978622695
Testing MAE:  2.9733567049538507

Training RMSE:  3.8693137416946244
Testing RMSE:  3.908834522345046
```

INFERENCE

> The MAE and RMSE for the testing data are also quite high, which means the model is not performing very well on unseen data. It may be necessary to further tune the model hyperparameters or try a different algorithm.

> The feature importance plot shows that the most important features for predicting life expectancy are `HIV/AIDS`, `income_composition_of_resources`, `adult_mortality`

VISUALIZATION OF FEATURE IMPORTANCE

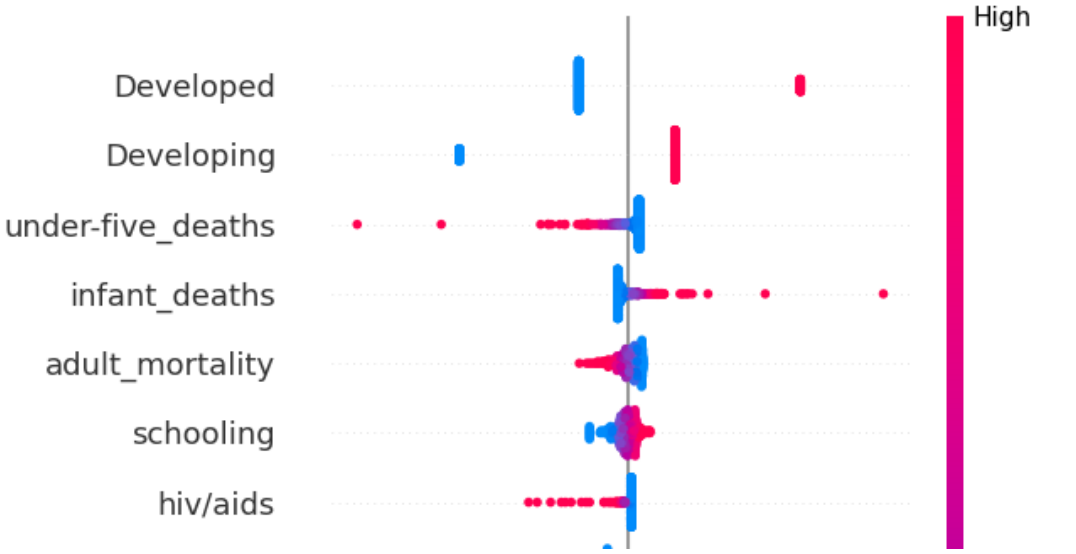
SHAP ANALYSIS - LINEAR REGRESSION

```
explainer = shap.Explainer(model.predict, X_train)
shap_values = explainer(X_test)
```

```
Permutation explainer: 484it [00:12,  6.26it/s]
```

```
#Summary plot
shap.summary_plot(shap_values, X_test)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be igr
```

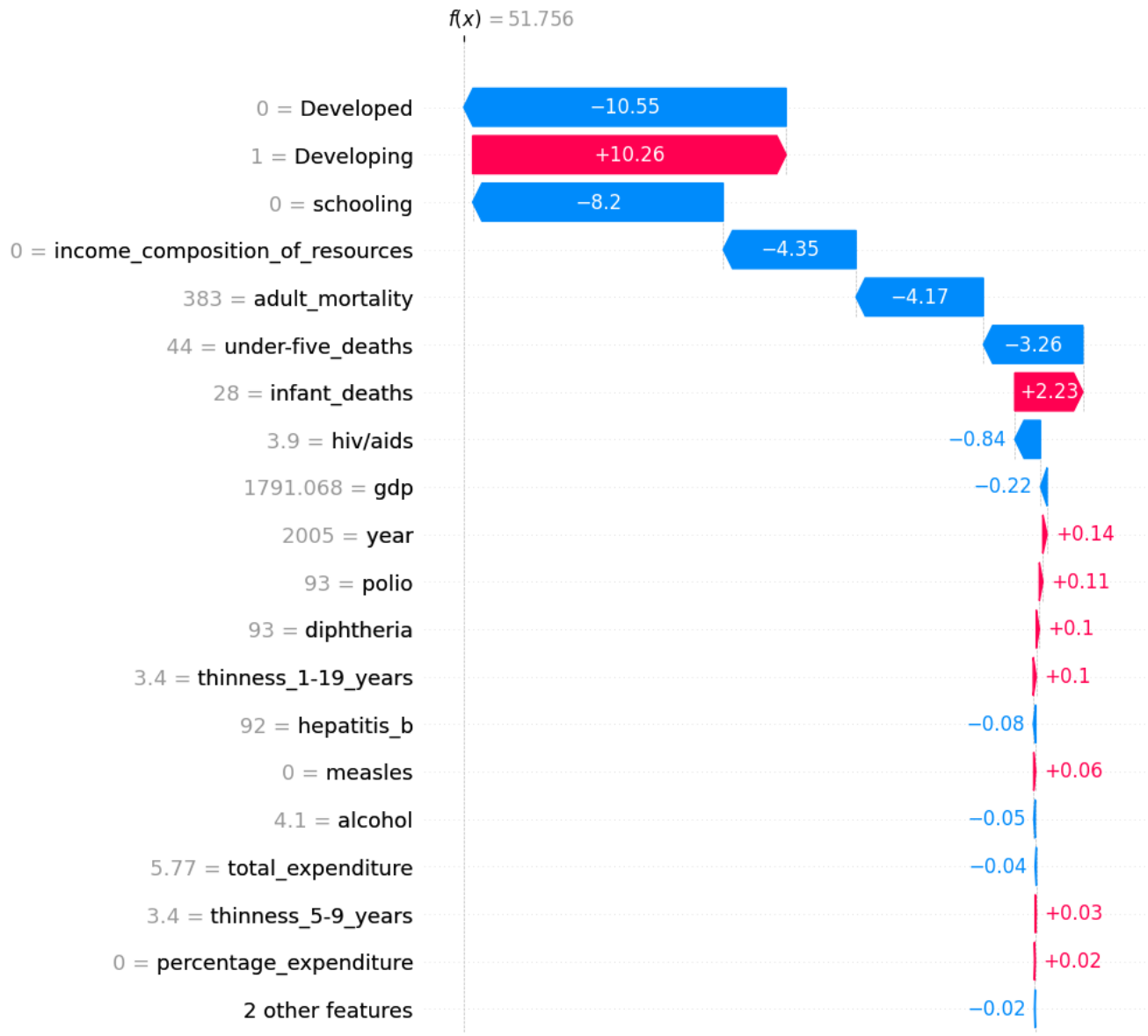


EXPLANATION : SHAP SUMMARY

> High values of `Hiv/aids`,`under-five_deaths` have a **negative** impact on the outcome anf `infant_deaths` have a **positive** impact

---


```
#Waterfall Plot
shap.plots.waterfall(shap_values[0], max_display=20, show=True)
```



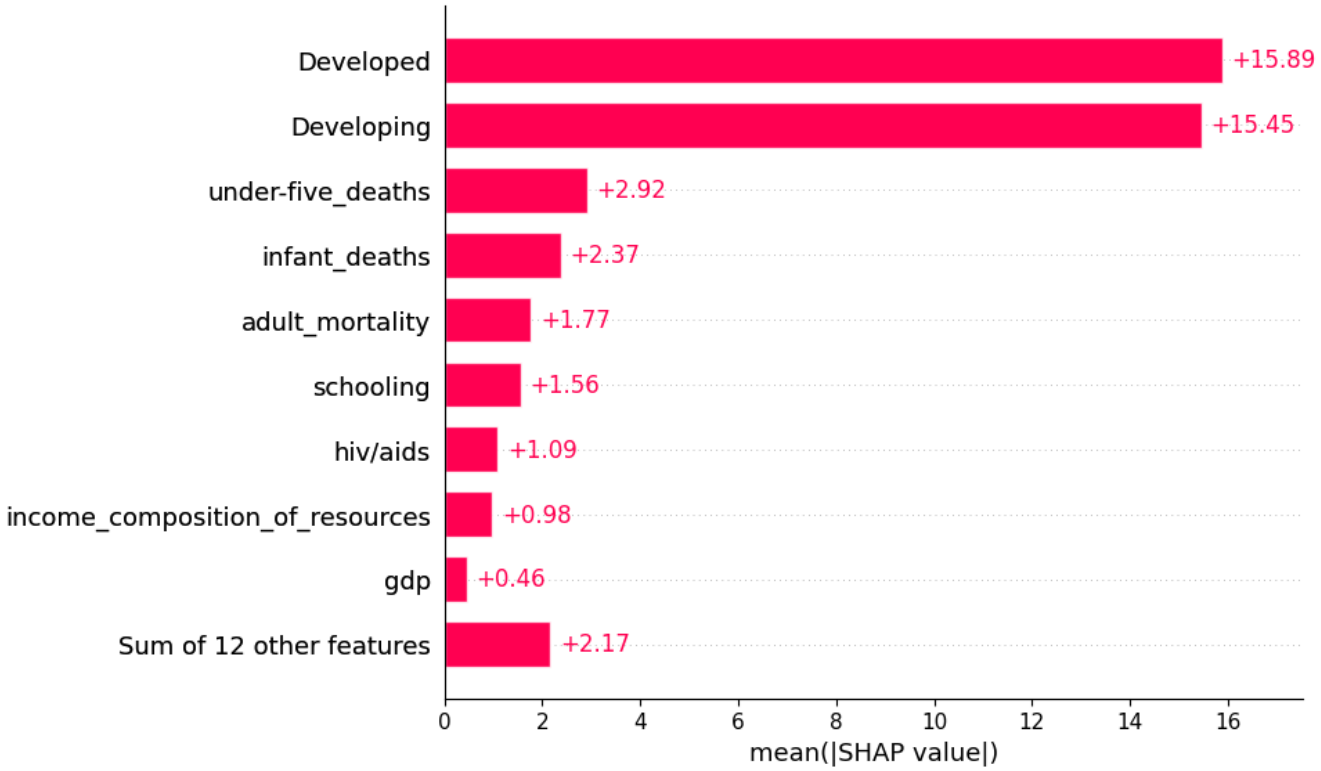
EXPLANATION : WATERFALL PLOT

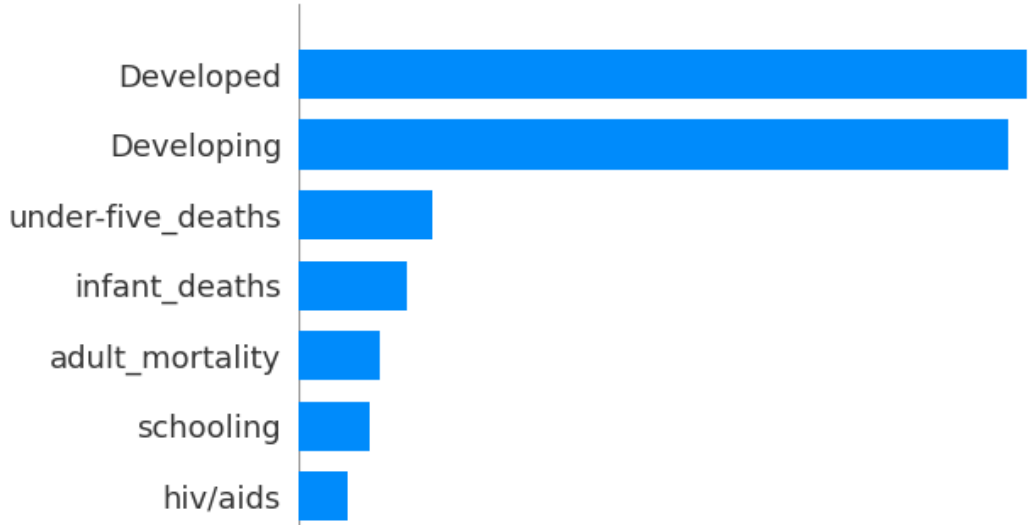
> E[f(x)] = 69.39 gives the average predicted life_expectancy. f(x) = 62.586 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

> SHAP values tell us how the features have contributed to the prediction when compared to the mean prediction. Large positive/negative values indicate that the feature had a significant impact on the model's prediction.

```
#Bar Plot
shap.plots.bar(shap_values)
```

```
#Bar Summary Plot
shap.summary_plot(shap_values, X_test, plot_type="bar")
```





EXPLANATION : BAR PLOT VS SUMMARY PLOT

> The **shap.plots.bar()**function generates a bar plot to display the importance of each feature in the dataset according to their Shapley values.

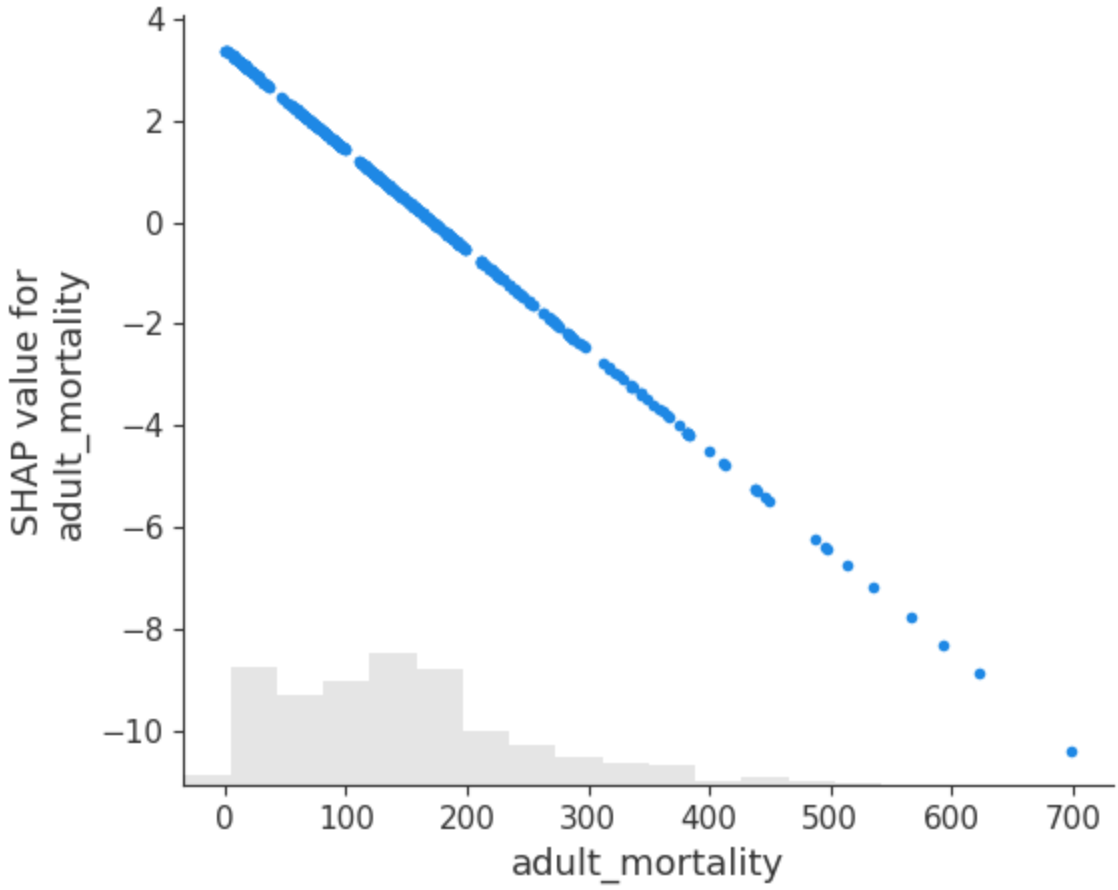
> **shap.plots.bar(shap_values)** generates a bar plot that shows the mean absolute SHAP value for each feature, averaged over all instances in the dataset. This plot provides a quick overview of the most important features and their direction of effect on the target variable.

> On the other hand, **shap.summary_plot(shap_values, X_test, plot_type="bar")** generates a more detailed bar plot that shows the distribution of SHAP values for each feature, along with their interaction effects. This plot provides a more granular view of the feature importance, as well as any potential interactions between features.

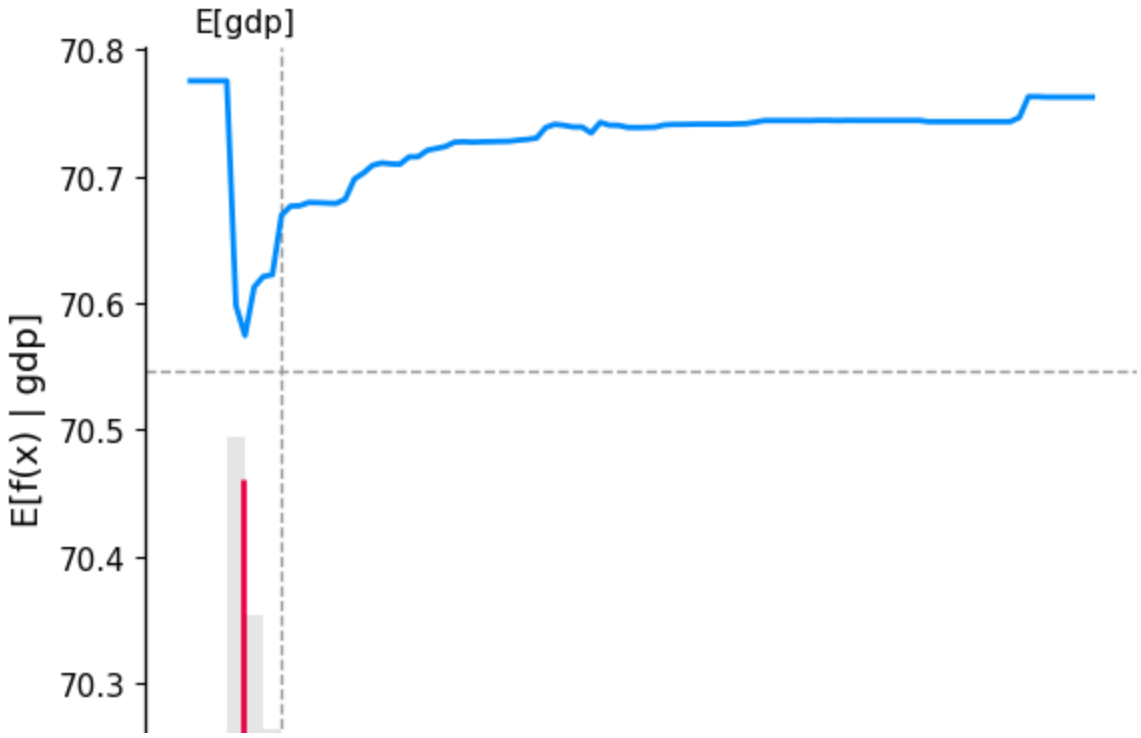
> In summary, shap.plots.bar(shap_values) is a quick way to visualize the most important features, while shap.summary_plot(shap_values, X_test, plot_type="bar") provides a more detailed view of the feature importance and interaction effects.

mean(|SHAP value|) (average impact on model output mag

```
#Plot 1: PDP
shap.plots.scatter(shap_values[:,"adult_mortality"])
```




```
partial_dependence_plot(rf,'gdp', 0)
```



EXPLANATION : PARTIAL DEPENDANCY PLOT

1. The dependency plot tells us that the relationship is not perfectly linear. There is a significant decrease in the SHAP value of Adult mortality as seen the graph.

2. bmi is seen to increase linearly as SHAP value increases.

XGBOOST

XGBoost (Extreme Gradient Boosting) is a popular machine learning algorithm for regression and classification problems. It is a type of gradient boosting method that builds an ensemble of decision trees to make predictions.

The key idea behind XGBoost is to sequentially add decision trees to the ensemble while minimizing a loss function, such as mean squared error, and regularizing the complexity of the trees to prevent overfitting. The algorithm optimizes the objective function by using gradient descent and second-order optimization techniques, which leads to faster and more accurate convergence compared to other gradient boosting algorithms.

Splitting the data into Train, Test and Split

```
# Split the data into training and testing sets
# Import necessary libraries

import xgboost as xgb

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state
```

Fitting the Model

```
# Fit the model using XGBoost
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, max_depth=4,
model.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.5, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

Model Prediction

```
prediction(X_train, X_test, y_train, y_test)
```

```
Training MAE:  1.1491452947428809
Testing MAE:  1.472642089022366

Training RMSE:  1.595047332655355
Testing RMSE:  2.042398787396814
```

INFERENCE

> The training MAE value of 1.15 suggests that on average, your model's predictions have an error of approximately 1.15 units compared to the actual values in the training dataset. Similarly, the testing MAE value of 1.47 indicates that the average prediction error is around 1.47 units in the testing dataset.

> The training RMSE value of 1.60 suggests that the average squared error of your model's predictions is approximately 1.60 units in the training dataset. Similarly, the testing RMSE value of 2.10 indicates that the average squared error is around 2.10 units in the testing dataset

---

SHAP ANALYSIS - LINEAR REGRESSION

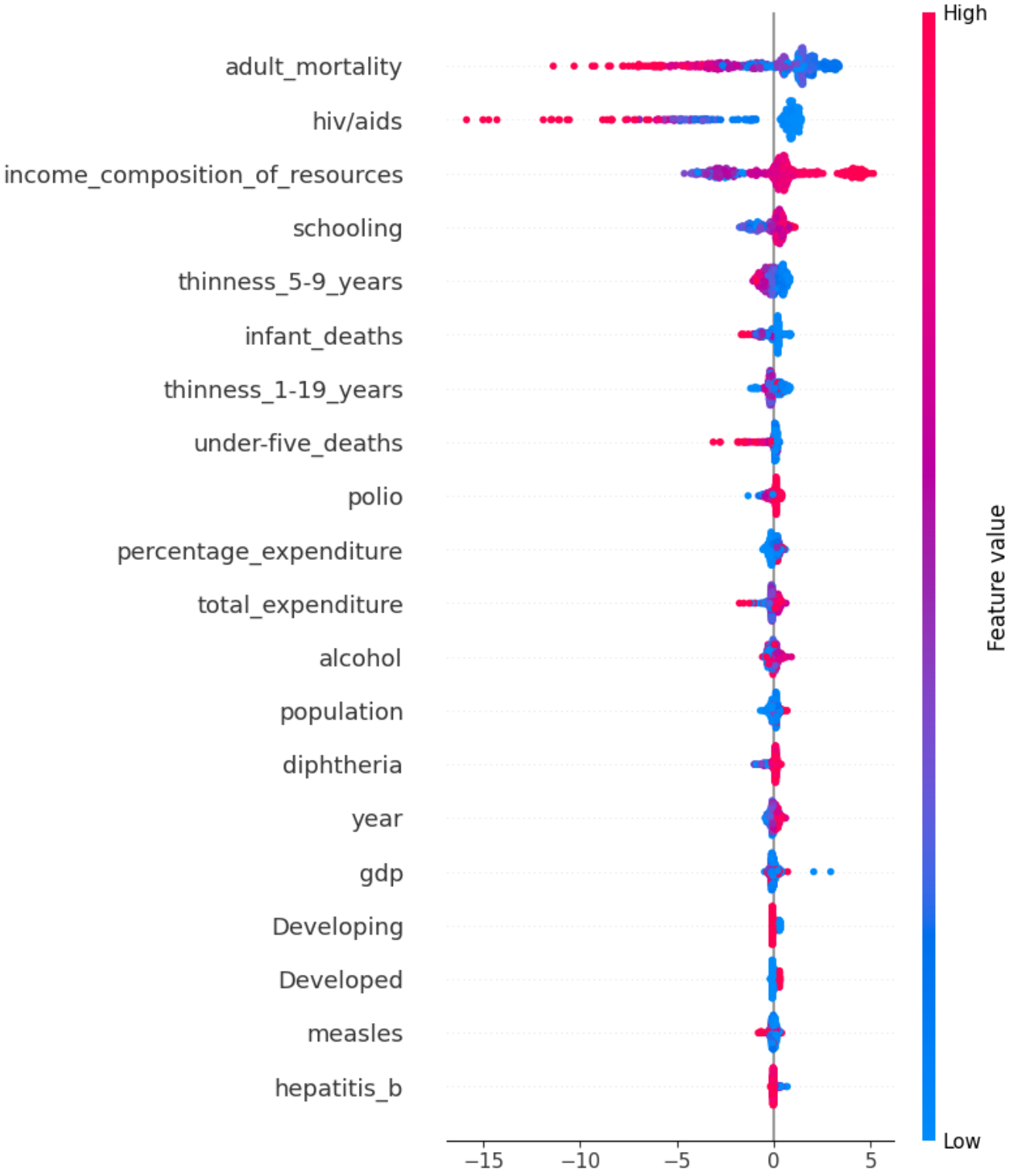
> SHAP (SHapley Additive exPlanations) analysis can be used to interpret the output of a linear regression model by providing explanations for individual predictions. It can help identify which features are contributing the most to the predicted output, and how each feature affects the output.

```
# Create an explainer object for the trained model using SHAP library
explainer = shap.Explainer(model.predict, X_train)

# Calculate SHAP values for the test data using the explainer object
shap_values = explainer(X_test)
```

```
Permutation explainer: 484it [01:27,  5.09it/s]
```

```
#Summary plot
shap.summary_plot(shap_values, X_test)
```

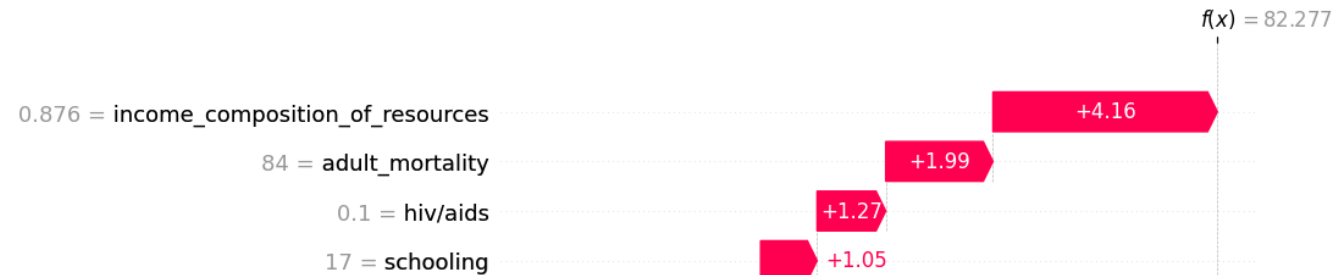
```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be igr
```



EXPLANATION : SHAP SUMMARY

> High values of `adult_mortality`,`hiv/aids` have a **negative** impact on the outcome anf `income_composition_of_resourses` have a **positive** impact

```
#Waterfall Plot
shap.plots.waterfall(shap_values[0], max_display=20, show=True)
```

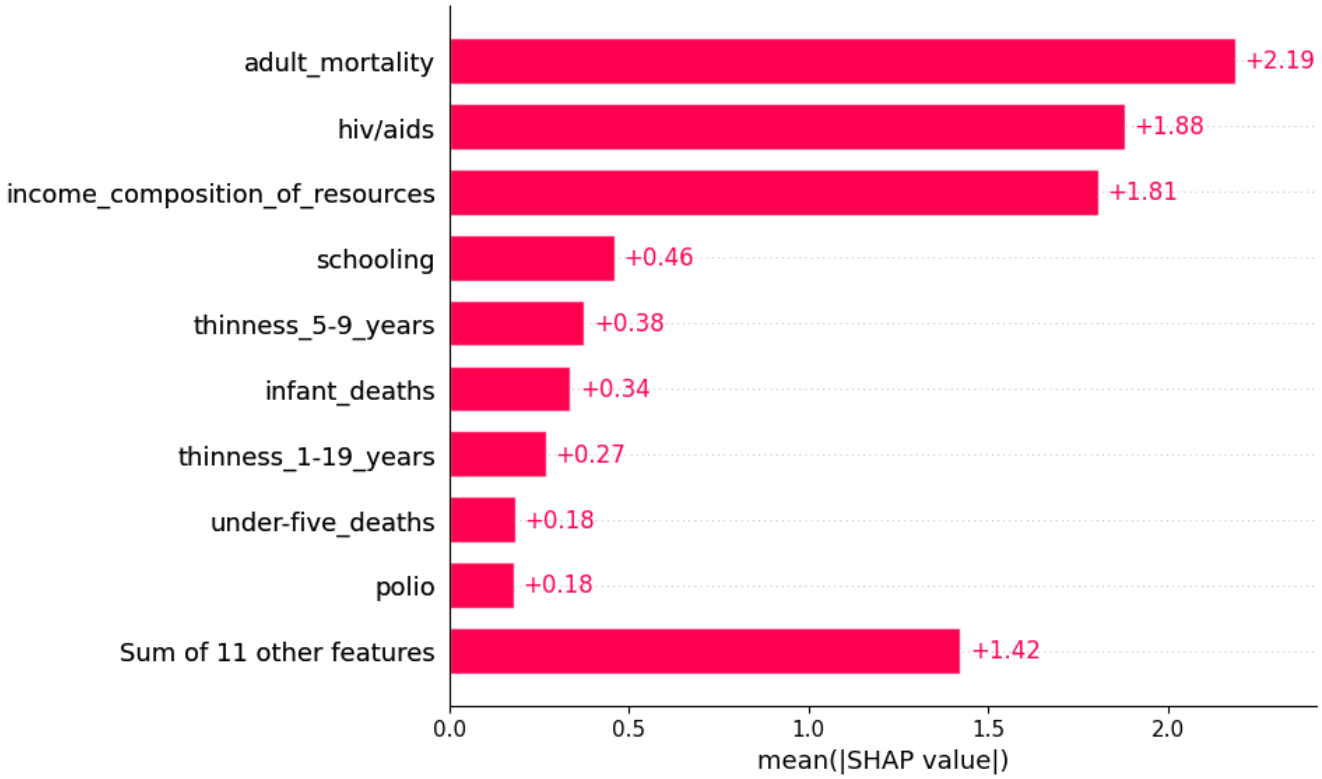


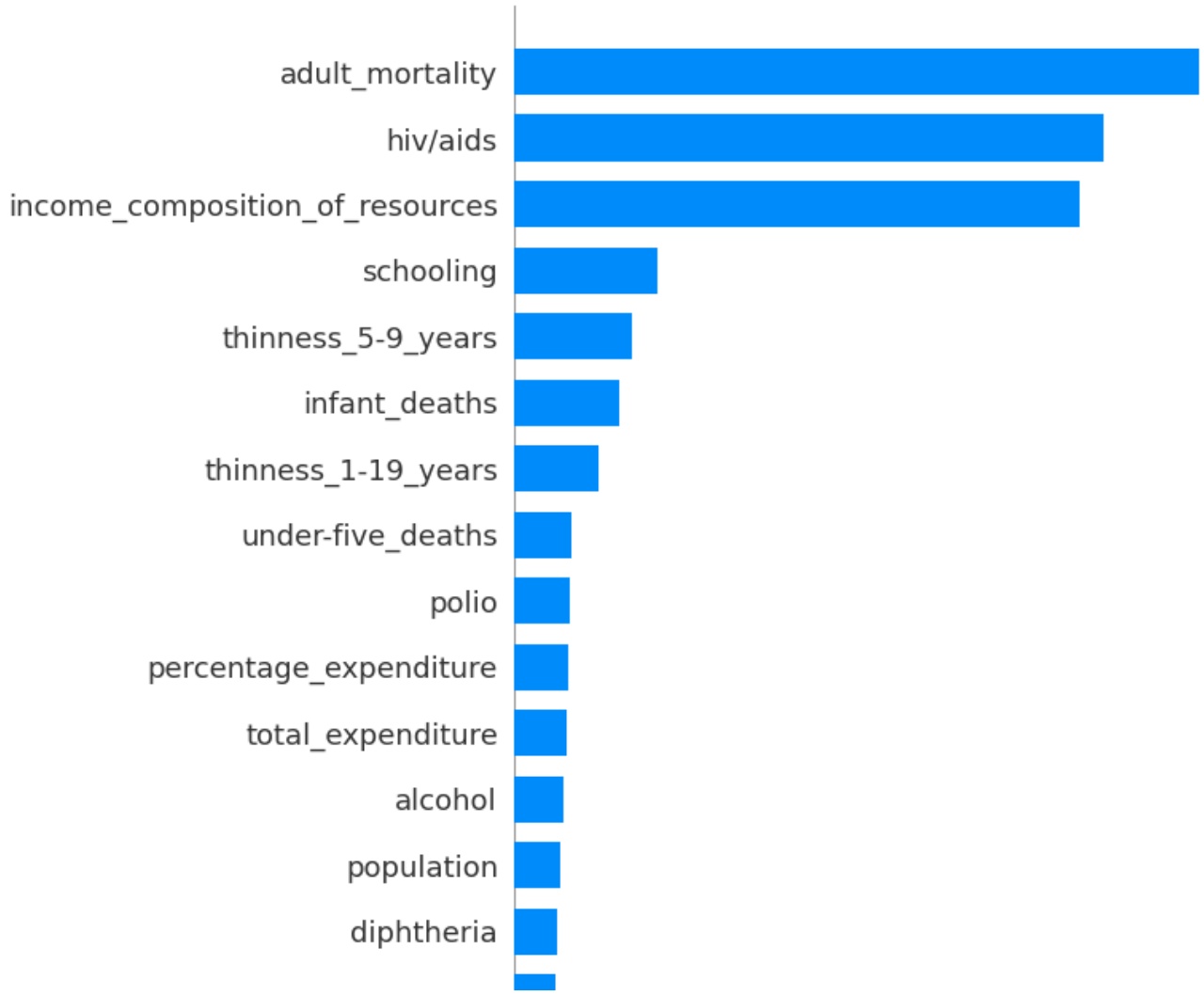
EXPLANATION : WATERFALL PLOT

> E[f(x)] = 69.155 gives the average predicted life_expectancy. f(x) = 82.291 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

> SHAP values tell us how the features have contributed to the prediction when compared to the mean prediction. Large positive/negative values indicate that the feature had a significant impact on the model's prediction.


```
#Bar Plot
shap.plots.bar(shap_values)


#Bar Summary Plot
shap.summary_plot(shap_values, X_test, plot_type="bar")
```


adult_mortality
hiv/aids
income_composition_of_resources
schooling
thinness_5-9_years
infant_deaths
thinness_1-19_years
under-five_deaths
polio
Sum of 11 other features
+2.19
+1.88
+1.81
+0.46
+0.38
+0.34
+0.27
+0.18
+0.18
+1.42
0.0
0.5
1.0
1.5
2.0
mean(|SHAP value|)


adult_mortality
hiv/aids
income_composition_of_resources
schooling
thinness_5-9_years
infant_deaths
thinness_1-19_years
under-five_deaths
polio
percentage_expenditure
total_expenditure
alcohol
population
diphtheria

EXPLANATION : BAR PLOT VS SUMMARY PLOT

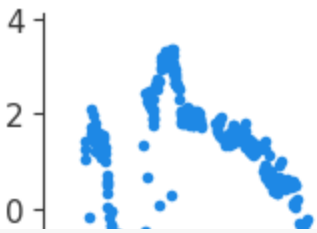
> The **shap.plots.bar()**function generates a bar plot to display the importance of each feature in the dataset according to their Shapley values.

> **shap.plots.bar(shap_values)** generates a bar plot that shows the mean absolute SHAP value for each feature, averaged over all instances in the dataset. This plot provides a quick overview of the most important features and their direction of effect on the target variable.

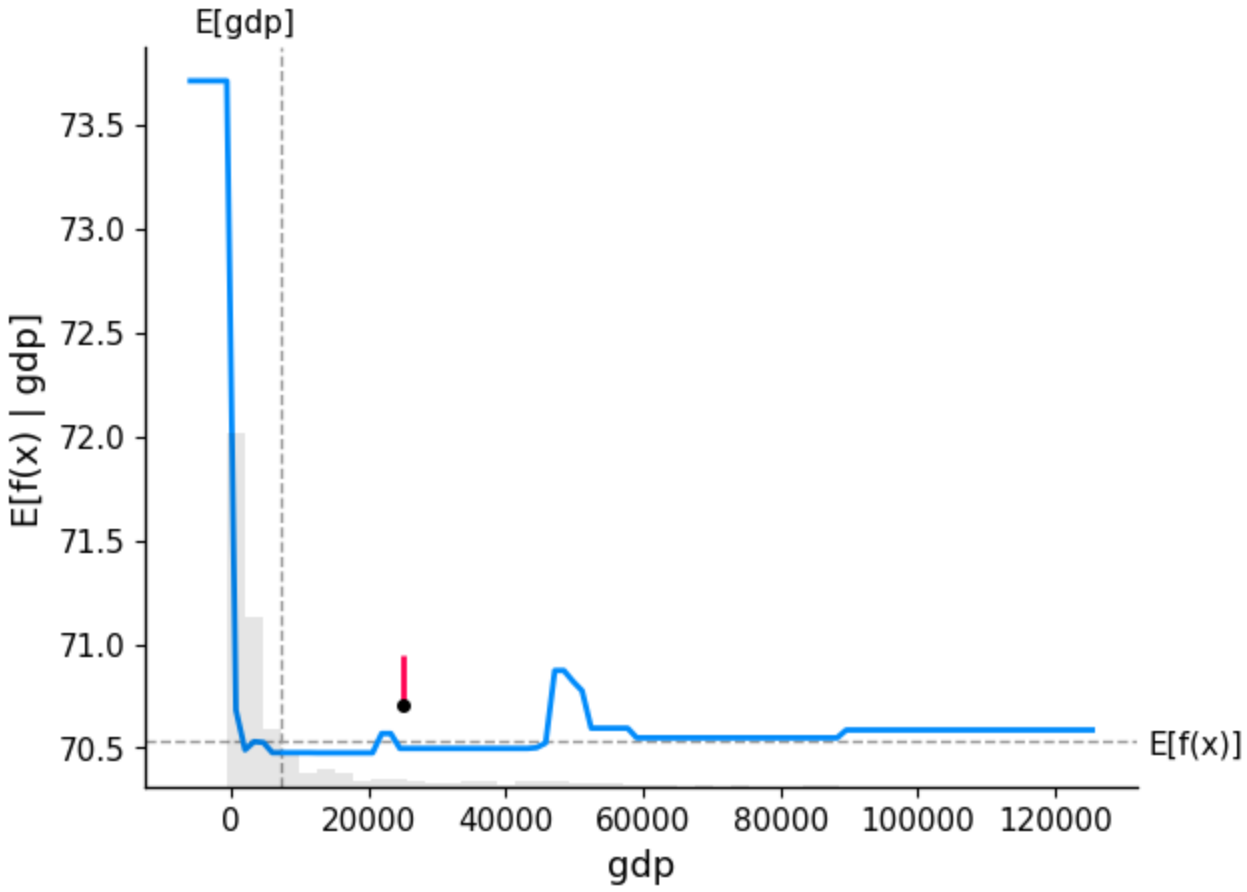
> On the other hand, **shap.summary_plot(shap_values, X_test, plot_type="bar")** generates a more detailed bar plot that shows the distribution of SHAP values for each feature, along with their interaction effects. This plot provides a more granular view of the feature importance, as well as any potential interactions between features.

> In summary, shap.plots.bar(shap_values) is a quick way to visualize the most important features, while shap.summary_plot(shap_values, X_test, plot_type="bar") provides a more detailed view of the feature importance and interaction effects.

```
#Plot 1: PDP
shap.plots.scatter(shap_values[:,"adult_mortality"])
```




```
partial_dependence_plot(model,'gdp', 0)
```



EXPLANATION : PARTIAL DEPENDANCY PLOT

---

1. The dependency plot tells us that the relationship is not perfectly linear. There is a significant decrease in the SHAP value of Adult mortality as seen the graph.

2. gdp is seen to increase linearly as SHAP value increases.

Use auto ml to find the best model

```
#convert pandas df into h2o frame
#initiate
h2o.init()
h2o_df = h2o.H2OFrame(df_life_expectancy)

#preview
h2o_df
```

```
Checking whether there is an H2O instance running at http://localhost:54321. cor
```

H2O_cluster_uptime:	13 mins 37 secs
H2O_cluster_timezone:	Etc/UTC
H2O_data_parsing_timezone:	UTC

```
#statistics
h2o_df.describe()
```

```
Rows:2413
Cols:38
```

	country	year	status	life_expectancy	adult_mortality	
type	enum	int	enum	real	int	
mins		2000.0		36.3	1.0	
mean		2007.80356402818		70.51193535018629	155.2436800663075	1
maxs		2015.0		89.0	723.0	
sigma		4.524618917340964		8.970740078426696	119.3523372054428	2
zeros		0		0	0	
missing	0	0	0	0	0	
0	Albania	2015.0	Developing	77.8	74.0	
1	Algeria	2015.0	Developing	75.6	19.0	
2	Angola	2015.0	Developing	52.4	335.0	
3	Antigua and Barbuda	2015.0	Developing	76.4	13.0	
4	Argentina	2015.0	Developing	76.3	116.0	
5	Armenia	2015.0	Developing	74.8	118.0	
6	Australia	2015.0	Developed	82.8	59.0	
7	Austria	2015.0	Developed	81.5	65.0	
8	Azerbaijan	2015.0	Developing	72.7	118.0	
9	Bahamas	2015.0	Developing	76.1	147.0	

```
[2413 rows x 38 columns]
```

```
#split train and test sets
train, test = h2o_df.split_frame(ratios=[.80])
```

```
#initiate

aml = H2OAutoML(stopping_metric='RMSE',     #for regression
                seed=121,
                max_models=10)

#train
aml.train(list(X.columns),
          y = 'life_expectancy',
          training_frame=train)
```

```
AutoML progress: |████████████████████████████████████████████████████████████
```

```
Model Details
=============
H2OStackedEnsembleEstimator : Stacked Ensemble
Model Key: StackedEnsemble_AllModels_1_AutoML_2_20230424_133923
```

Model Summary for Stacked Ensemble:

key	value
Stacking strategy	cross_validation
Number of base models (used / total)	8/10
# GBM base models (used / total)	3/4
# XGBoost base models (used / total)	3/3
# DRF base models (used / total)	2/2
# GLM base models (used / total)	0/1
Metalearner algorithm	GLM
Metalearner fold assignment scheme	Random
Metalearner nfolds	5
Metalearner fold_column	None
Custom metalearner hyperparameters	None

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on train data. **

MSE: 0.5314902887752537
RMSE: 0.7290338049605476
MAE: 0.45454697257171767
RMSLE: 0.010698475658860606
Mean Residual Deviance: 0.5314902887752537
R^2: 0.9935857812724215
Null degrees of freedom: 1945
Residual degrees of freedom: 1937
Null deviance: 161248.02503378005
```

LEADERBOARD EXPLORATION

---

1. The aml object has a dashboard called the leaderboard that shows all the models trained in the process, along with their scores and metrics for ranking the models. For example, in our multivariate regression case, the metric used is MRSE.

2. The leaderboard displays the models with their corresponding metrics. When given an H2OAutoML object, the leaderboard shows 5-fold cross-validated metrics by default.

AIC: 7869.059000810119

```
#leaderboard

lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

model_id	rmse	mse	mae	rm
StackedEnsemble_AllModels_1_AutoML_2_20230424_133923	1.82744	3.33954	1.15902	0.027
StackedEnsemble_BestOfFamily_1_AutoML_2_20230424_133923	1.83455	3.36559	1.17215	0.027
GBM_4_AutoML_2_20230424_133923	1.84909	3.41914	1.18645	0.027
GBM_3_AutoML_2_20230424_133923	1.89857	3.60457	1.23911	0.028
GBM_2_AutoML_2_20230424_133923	1.91058	3.6503	1.24716	0.028
DRF_1_AutoML_2_20230424_133923	1.96018	3.84231	1.24702	0.030
XRT_1_AutoML_2_20230424_133923	1.97581	3.90384	1.25572	0.030
XGBoost_3_AutoML_2_20230424_133923	2.15127	4.62794	1.47964	0.032
XGBoost_1_AutoML_2_20230424_133923	2.20694	4.87059	1.49112	0.033
XGBoost_2_AutoML_2_20230424_133923	2.23001	4.97294	1.53344	0.033
GBM_1_AutoML_2_20230424_133923	2.498	6.24001	1.73435	0.039
GLM_1_AutoML_2_20230424_133923	4.03564	16.2864	2.95705	0.065

[12 rows x 6 columns]

INFERENCE

---

1. The models are ranked based on their performance measures, including `mean_residual_deviance`, `rmse`, `mse`, `mae`, and `rmsle`.

2. The models with smaller error values are considered to be a better fit and have higher performance.

3. The top-performing model are:
    1. **StackedEnsemble_AllModels_1_AutoML_1_20230225_03829**
    2. second-best model, **StackedEnsemble_BestOfFamily_1_AutoML_1_20230225_03829**
    3. third best performing is **GBM_4_AutoML_1_20230325_143233**

4. The All Models ensemble includes all the models, while the Best of Family ensemble includes the best-performing model from each algorithm class or family.

MODEL EXPLAINABILITY IN AUTOML H2O

1. H2O offers ways to explain how machine learning models work for both groups of models and individual models.
2. Model explainability is vital in machine learning because evaluating a model's accuracy alone is not enough to understand how the model makes decisions.
3. We must also comprehend how the model input variables function and how the model's predictions are affected when we alter the input variable values.

When h2o.explain() is provided with a list of models, the following global explanations will be generated by default:

1. Residual Analysis for Leader Model
2. Variable Importance of Top Base (non-Stacked) Model
3. Variable Importance Heatmap (compare all non-Stacked models)
4. Model Correlation Heatmap (compare all models)
5. SHAP Summary of Top Tree-based Model (TreeSHAP)
6. Partial Dependence (PD) Multi Plots (compare all models)
7. Individual Conditional Expectation (ICE) Plots

```
#compare all models
exm = aml.explain(test)
```

Leaderboard

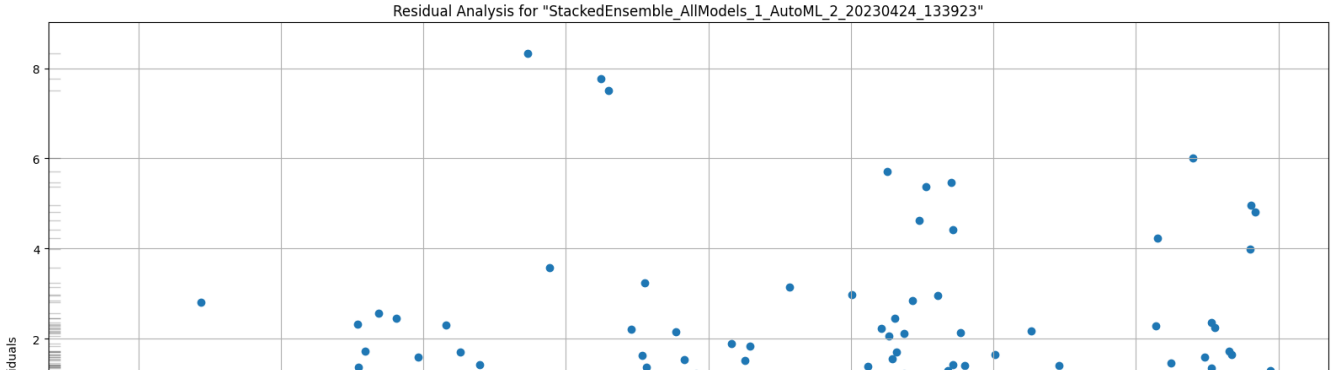
Leaderboard shows models with their metrics. When provided with H2OAutoML object, the leaderboard shows 5-fold cross-validated metrics by default (depending on the H2OAutoML settings), otherwise it shows metrics computed on the frame. At most 20 models are shown by default.

model_id	rmse	mse	mae	rm
StackedEnsemble_AllModels_1_AutoML_2_20230424_133923	1.546	2.39013	1.02654	0.022
StackedEnsemble_BestOfFamily_1_AutoML_2_20230424_133923	1.55115	2.40607	1.03214	0.02
GBM_4_AutoML_2_20230424_133923	1.55347	2.41327	1.05187	0.022
DRF_1_AutoML_2_20230424_133923	1.63366	2.66885	1.07004	0.024
GBM_3_AutoML_2_20230424_133923	1.64887	2.71879	1.09572	0.023
GBM_2_AutoML_2_20230424_133923	1.65928	2.75321	1.12462	0.024
XRT_1_AutoML_2_20230424_133923	1.68165	2.82796	1.11195	0.024
XGBoost_1_AutoML_2_20230424_133923	1.88545	3.55492	1.38163	0.027
XGBoost_3_AutoML_2_20230424_133923	1.88835	3.56585	1.32135	0.027
XGBoost_2_AutoML_2_20230424_133923	2.06007	4.24388	1.41937	0.030
GBM_1_AutoML_2_20230424_133923	2.13744	4.56866	1.55256	0.032
GLM_1_AutoML_2_20230424_133923	3.72042	13.8415	2.77859	0.055

[12 rows x 9 columns]

Residual Analysis

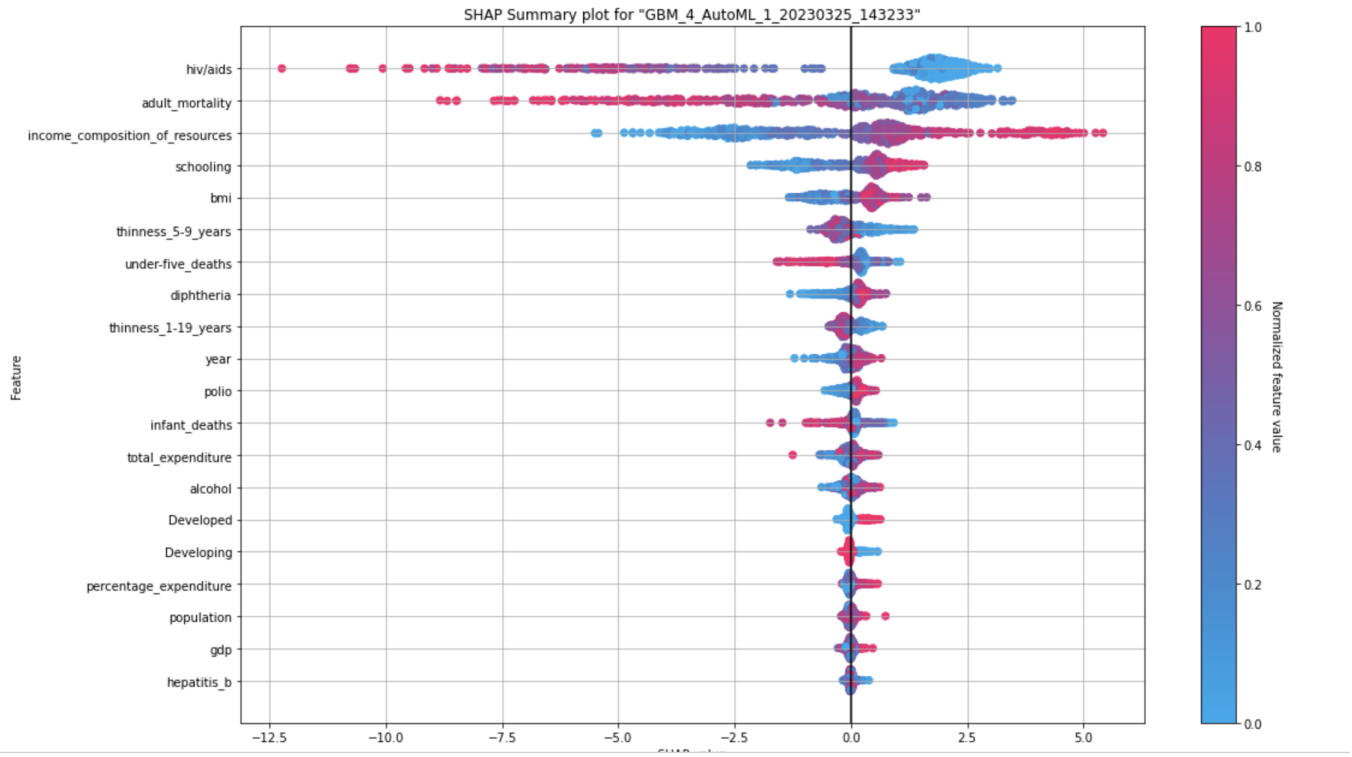
Residual Analysis plots the fitted values vs residuals on a test dataset. Ideally, residuals should be randomly distributed. Patterns in this plot can indicate potential problems with the model selection, e.g., using simpler model than necessary, not accounting for heteroscedasticity, autocorrelation, etc. Note that if you see "striped" lines of residuals, that is an artifact of having an integer valued (vs a real valued) response variable.



EXPLANATION : SHAP SUMMARY AND PDP

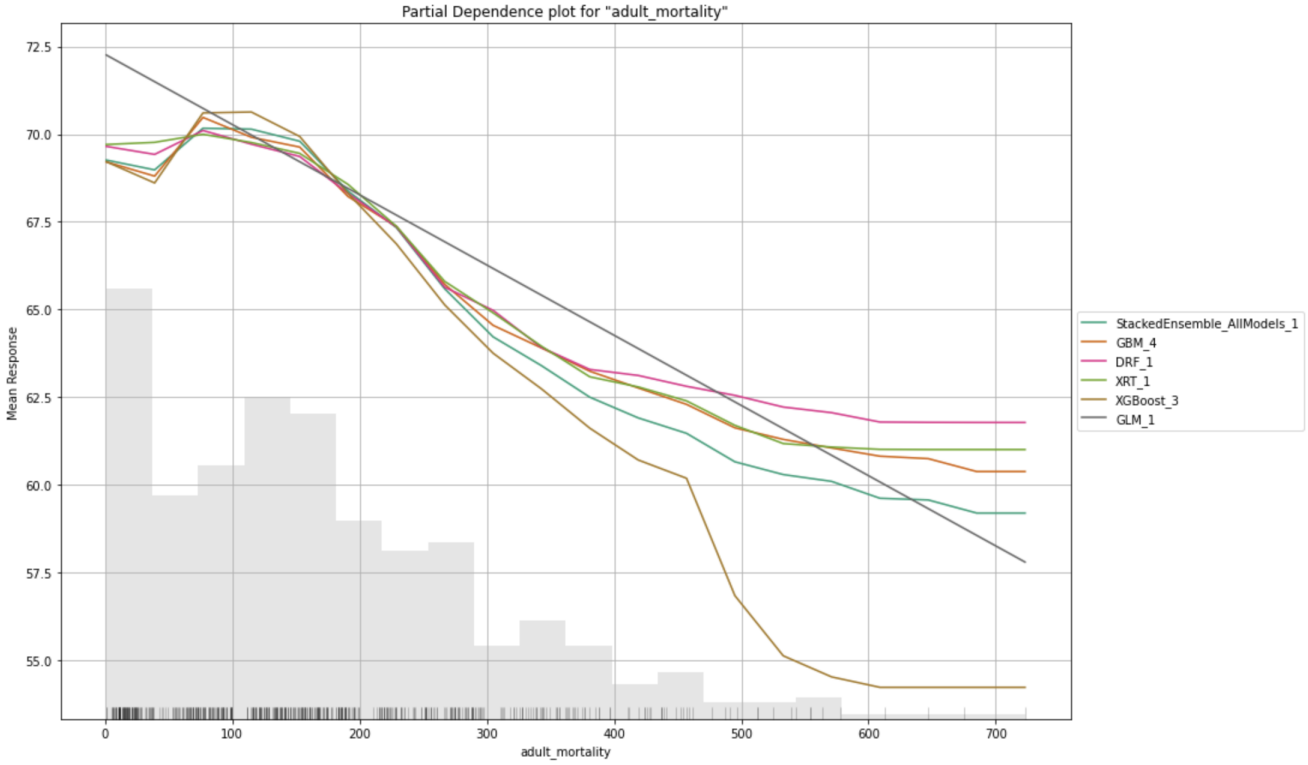
SHAP SUMMARY

SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the prediction before applying inverse link function.



1. High values of `Hiv/aids` and `adult_mortality` have a **negative** impact on the outcome, while high values of `income_composition_of_resources` and `schooling` have a **positive** impact.

2. Other features such as `thinness`, `BMI`, `under-five_deaths`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.



INFERENCE 1 : `adult_mortality`

> The models accurately identified the **inverse relationship** between `adult_mortality` and `life_expectancy`, meaning that when adult mortality rates are high, life expectancy is lower.

NEURAL NETWORKS


```
from keras.models import Sequential
from keras.layers import Dense
```


```
model = Sequential()
model.add(Dense(10, input_dim=20, activation='tanh'))

model.add(Dense(1, activation='sigmoid'))
```

```
len(X.columns)
```

```
20
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state
```

```
x_train_keras = np.array(X_train)
y_train_keras = np.array(y_train)
y_train_keras = y_train_keras.reshape(y_train_keras.shape[0], 1)
```

```
model.fit(np.array(x_train_keras), np.array(y_train_keras), epochs=10, batch_size=128
```

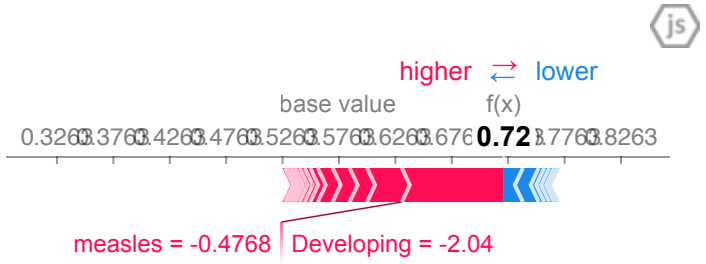
```
Epoch 1/10
16/16 [==============================] - 1s 3ms/step - loss: 94.4956 - accuracy:
Epoch 2/10
16/16 [==============================] - 0s 4ms/step - loss: 82.3362 - accuracy:
Epoch 3/10
16/16 [==============================] - 0s 2ms/step - loss: 70.2453 - accuracy:
Epoch 4/10
16/16 [==============================] - 0s 2ms/step - loss: 58.1931 - accuracy:
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 46.0918 - accuracy:
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 34.0713 - accuracy:
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 22.0405 - accuracy:
Epoch 8/10
16/16 [==============================] - 0s 5ms/step - loss: 10.0063 - accuracy:
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: -1.9597 - accuracy:
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: -13.9202 - accuracy
<keras.callbacks.History at 0x7f5228ea6310>
```

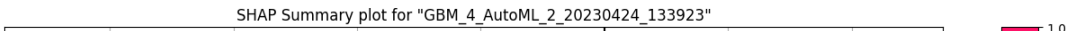
```
scores = model.evaluate(np.array(X_test), np.array(y_test))
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
16/16 [==============================] - 0s 3ms/step - loss: -20.2618 - accuracy

accuracy: 0.00%
```

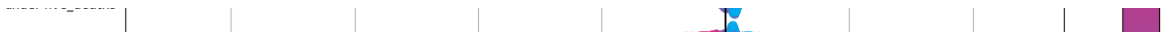
```
# SHAP Analysis
from sklearn.preprocessing import StandardScaler

features = list(X.columns)
scaler = StandardScaler().fit(X_train[features])
```

```
X_train[features] = scaler.transform(X_train[features])
X_test[features] = scaler.transform(X_test[features])

explainer = shap.KernelExplainer(model, X_train.iloc[:50,:])
shap_values = explainer.shap_values(X_train.iloc[20,:], nsamples=500)

shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[0], X_train.iloc[20,:])
```





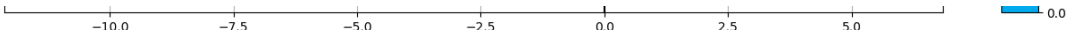
SHAP analysis on the models from steps 1, 2, and 3, interpret the SHAP values and compare them with the other model interpretability methods.



The SHAP Analysis has been done above for:

1. Linear Regression
2. XGBoost
3. Random Forest
4. AutoML

COMPARISON OF SHAP RESULTS AND PDP AND LIME



LINEAR REGRESSION

1. High values of `Hiv/aids` and `under-five_deaths` have a **negative** impact on the outcome.

2. Other features such as `thinness`, `BMI`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.

3. E[f(x)] = 69.39 gives the average predicted life_expectancy. f(x) = 62.586 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

XGBOOST

---

1. High values of `adult_mortality`,`hiv/aids` have a **negative** impact on the outcome and `income_composition_of_resourses` have a **positive** impact

2. E[f(x)] = 69.155 gives the average predicted life_expectancy. f(x) = 82.291 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

RANDOM FOREST

---

1. High values of `Hiv/aids`,`under-five_deaths` have a **negative** impact on the outcome anf `infant_deaths` have a **positive** impact

2. E[f(x)] = 69.39 gives the average predicted life_expectancy. f(x) = 62.586 is the predicted number of rings for this particular abalone. The SHAP values are all the values in between.

AUTOML

The top-performing model are:

1. **StackedEnsemble_AllModels_1_AutoML_1_20230225_03829**
2. second-best model, **StackedEnsemble_BestOfFamily_1_AutoML_1_20230225_03829**
3. third best performing is **GBM_4_AutoML_1_20230325_143233**

---

1. High values of `Hiv/aids` and `adult_mortality` have a **negative** impact on the outcome, while high values of `income_composition_of_resources` and `schooling` have a **positive** impact.

2. Other features such as `thinness`, `BMI`, `under-five_deaths`, `year`, `alcohol`, `polio`, `diphtheria`, `total_expenditure`, and others have little impact on the outcome.

Implementing and Comparing model interpretability methods like LIME and Partial Dependence Plot Analysis

---

```
!pip install lime
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 6.0 MB/s eta 0:00:0
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.9/di
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.9/di
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.9/dis
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/loc
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.9/d
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/pyth
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dis
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-package
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283859 sha
  Stored in directory: /root/.cache/pip/wheels/ed/d7/c9/5a0130d06d6310bc6cbe5522
Successfully built lime
```

```
Installing collected packages: lime
Successfully installed lime-0.2.0.1
```

```
import lime
import lime.lime_tabular
```

```
x = ['year',
 'adult_mortality',
 'infant_deaths',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'measles',
 'under-five_deaths',
 'polio',
 'total_expenditure',
 'diphtheria',
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',
 'schooling',
 'Developed',
 'Developing']

y = ['life_expectancy']
```


```
X = df_life_expectancy[x]
X.info()

Y = df_life_expectancy[y]
Y.head()

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
Data columns (total 20 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   year                             2413 non-null   int64  
 1   adult_mortality                  2413 non-null   float64
 2   infant_deaths                    2413 non-null   int64  
 3   alcohol                          2413 non-null   float64
 4   percentage_expenditure           2413 non-null   float64
 5   hepatitis_b                      2413 non-null   float64
```

```
 6   measles                          2413 non-null   int64
 7   under-five_deaths                2413 non-null   int64
 8   polio                            2413 non-null   float64
 9   total_expenditure                2413 non-null   float64
 10  diphtheria                       2413 non-null   float64
 11  hiv/aids                         2413 non-null   float64
 12  gdp                              2413 non-null   float64
 13  population                       2413 non-null   float64
 14  thinness_1-19_years              2413 non-null   float64
 15  thinness_5-9_years               2413 non-null   float64
 16  income_composition_of_resources  2413 non-null   float64
 17  schooling                        2413 non-null   float64
 18  Developed                        2413 non-null   uint8
 19  Developing                       2413 non-null   uint8
dtypes: float64(14), int64(4), uint8(2)
memory usage: 362.9 KB
```

LIME (Local Interpretable Model-Agnostic Explanations)

LIME (Local Interpretable Model-Agnostic Explanations) is a technique used in machine learning for interpreting the predictions made by complex models. It provides a way to explain the output of a machine learning model by approximating it with a simpler, interpretable model that is easier to understand.

LIME works by generating a dataset of "perturbed" examples around a specific instance that needs to be explained. These perturbed examples are then used to train a simpler, interpretable model, such as a decision tree or linear regression model. The coefficients of this simpler model are then used to explain the prediction of the original model.

In essence, LIME allows us to see which features of the input data were most influential in determining the model's prediction for a particular instance. This can be useful for understanding how a model is making its decisions, identifying potential biases or errors in the model, and building trust with stakeholders who may not have a technical background.

RandomForestRegressor

> RandomForestRegressor is a type of ensemble learning algorithm that is used for regression problems. It is a collection of decision trees, where each tree is trained on a random subset of features and a random subset of data points, and the final output is the average of the individual tree predictions.

> This technique helps to reduce overfitting and improve the generalization of the model. The RandomForestRegressor algorithm is widely used in various domains such as finance, healthcare, and marketing, where it is used to predict the values of continuous variables.

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

```
A column-vector y was passed when a 1d array was expected. Please change the sha
```

RandomForestRegressor

RandomForestRegressor(random_state=42)

Using LIME (Local Interpretable Model-Agnostic Explanations)

```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
feature_names=X_train.columns,
class_names=['Life expectancy'],
verbose=True,
mode='regression')
```

```
i = np.random.randint(0, X_test.shape[0])
```

```
# explain the prediction of the XGBoost model on the selected sample
exp = explainer.explain_instance(X_test.iloc[i], rf.predict, num_features=20)
print('Explanation for sample {}:'.format(i))
print(exp.as_list())
```

```
Intercept 62.82210966144413
Prediction_local [81.43217964]
Right: 79.91199999999994
Explanation for sample 417:
[('hiv/aids <= 0.10', 6.3405373002952485), ('income_composition_of_resources > (
X does not have valid feature names, but RandomForestRegressor was fitted with f
```

```
# explain the prediction of the XGBoost model on the selected sample
```