Predicting Life Expectancy and Health Insurance Linear Regression

Crash Course in Statistical Learning:Linear Regression Worked Examples

Shreya Jaiswal

ABSTRACT

This study aimed to predict life expectancy and health insurance charges using linear regression on a dataset of demographic, lifestyle, and medical factors. The dataset contained information on over 1300 individuals and included features such as age, BMI, smoking status, region, and other health indicators.

First, the data was preprocessed by removing missing values, encoding categorical variables, and scaling numerical features. Next, two separate linear regression models were trained to predict life expectancy and health insurance charges, respectively. The models were evaluated using metrics such as mean squared error, R-squared, and p-values to determine their accuracy and statistical significance.

Overall, the findings suggest that linear regression can be an effective method for predicting life expectancy and health insurance charges based on demographic and health-related factors. The study has implications for healthcare policy and insurance pricing, as well as for individuals seeking to understand and manage their health risks.

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: 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*

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

> ϵ = *the error term or residual*

STATISTICAL METRICS: MSE and VARIANCE

> 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 `scikit` inbuilt function.

WORKED EXAMPLE 1: WHO DATASET TO PREDICT LIFE EXPECTANCY USING LINEAR REGRESSION ALGORITHM

INTRODUCTION: LIFE EXPECTANCY

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

---

> **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

DATA DICTIONARY

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

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
Requirement already satisfied: Jinja2==3.0.0 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: MarkupSafe>=2.0.0rc2 in /usr/local/lib/python3.9/
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: eli5==0.12.0 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.9/di
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (fr
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: scipy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: MarkupSafe>=2.0.0rc2 in /usr/local/lib/python3.9/
```

```
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/
```

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

```
Requirement already satisfied: defusedxml in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.9/d
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: bleach in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.9/dist-package
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.9/
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.9/di
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
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
```

```
!pip install shap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: shap in /usr/local/lib/python3.9/dist-packages (0
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: numba in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/pyth
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
```

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 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 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

```
#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	
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	

```
df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   country                          2938 non-null   object 
 1   year                             2938 non-null   int64  
 2   status                           2938 non-null   object 
 3   life_expectancy                  2928 non-null   float64
 4   adult_mortality                  2928 non-null   float64
 5   infant_deaths                    2938 non-null   int64  
 6   alcohol                          2744 non-null   float64
 7   percentage_expenditure           2938 non-null   float64
 8   hepatitis_b                      2385 non-null   float64
 9   measles                          2938 non-null   int64  
 10  bmi                              2904 non-null   float64
 11  under-five_deaths                2938 non-null   int64  
 12  polio                            2919 non-null   float64
 13  total_expenditure                2712 non-null   float64
 14  diphtheria                       2919 non-null   float64
 15  hiv/aids                         2938 non-null   float64
 16  gdp                              2490 non-null   float64
 17  population                       2286 non-null   float64
 18  thinness_1-19_years              2904 non-null   float64
 19  thinness_5-9_years               2904 non-null   float64
 20  income_composition_of_resources  2771 non-null   float64
 21  schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

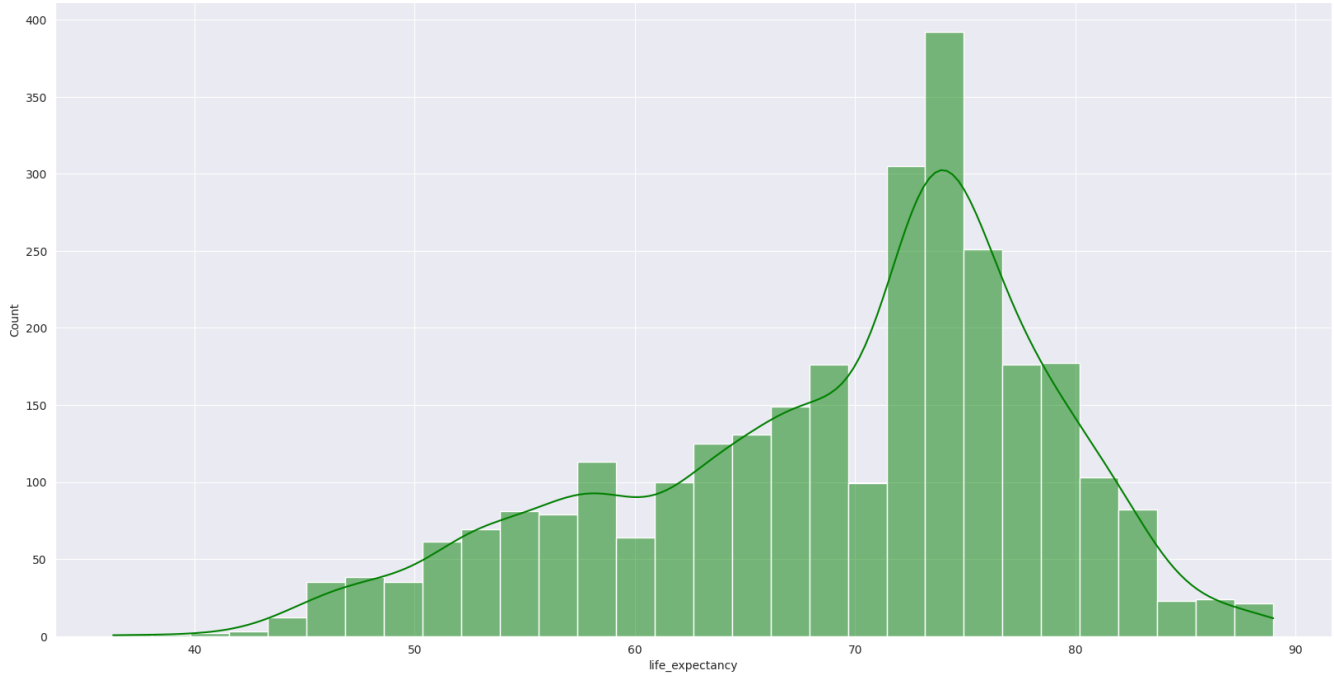
EXPLORATORY DATA ANALYSIS (EDA)

> 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

```
#PLOTTING THE HEATMAP
plt.figure(figsize=(20,10))
```

```
sns.heatmap(df_original.corr(), annot=True, cmap="mako")
```

```
<ipython-input-453-6570672d3273>:3: FutureWarning:

The default value of numeric only in DataFrame.corr is deprecated. In a future v
```

INFERENCE

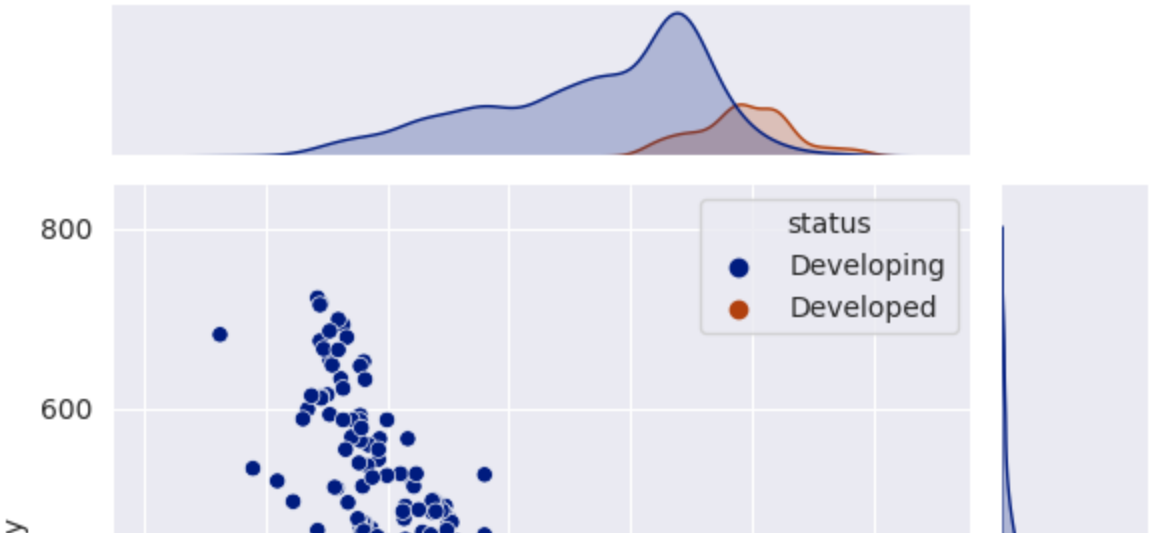
Insights obtained from the correlation matrix

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>
```



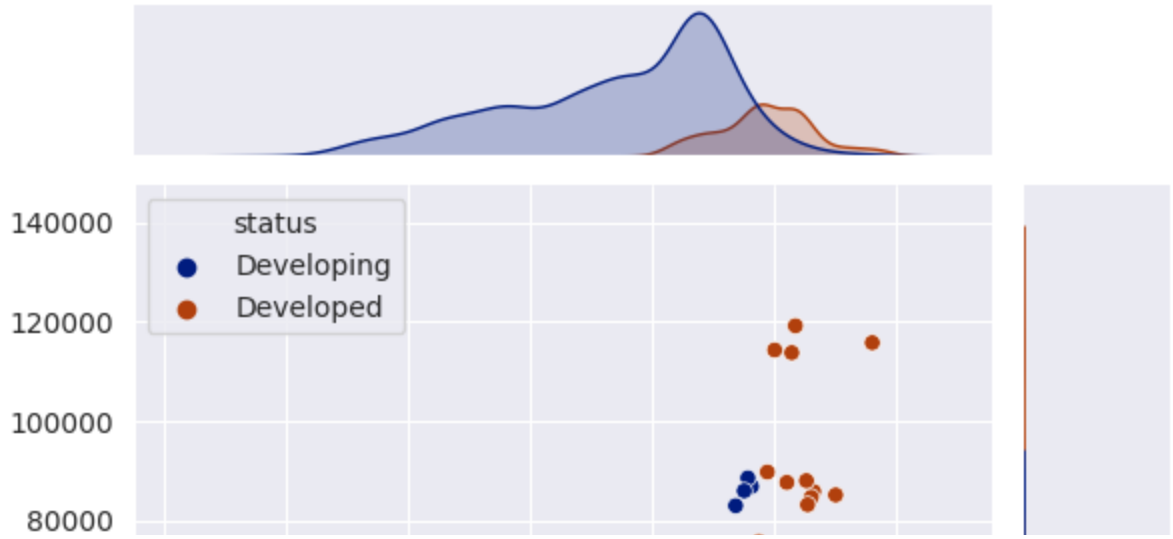
INFERENCE

> 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

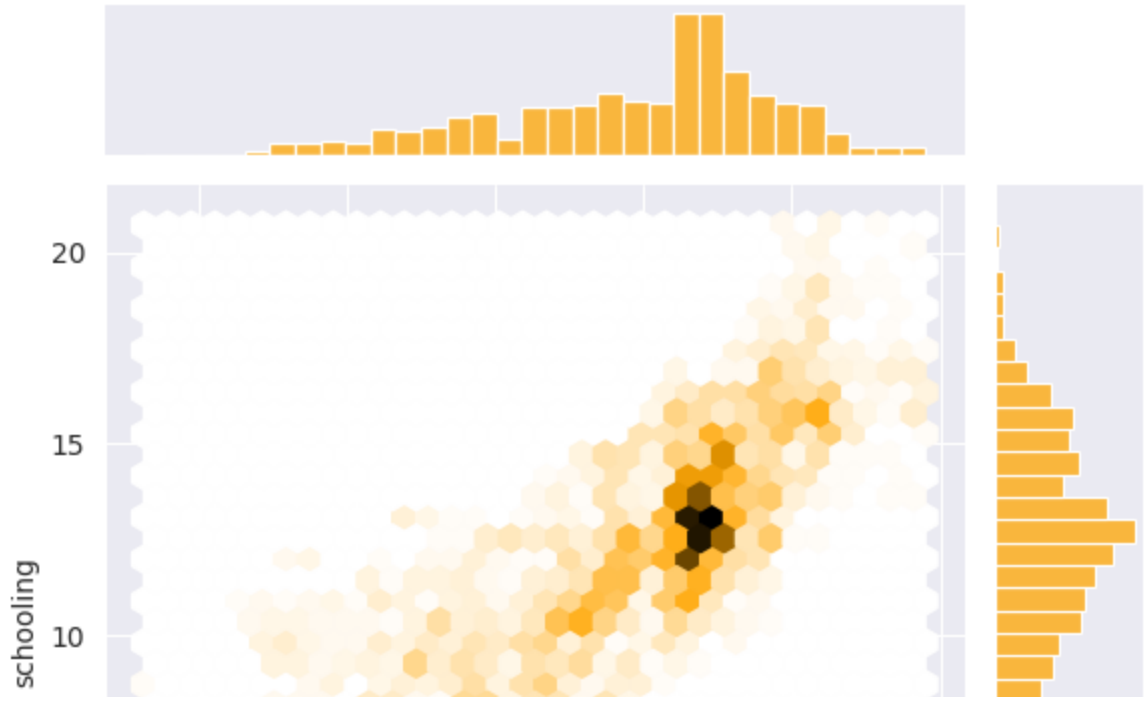
Plotting Life Expectance vs Schooling using hex plot

> 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.

life expectancy

```
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

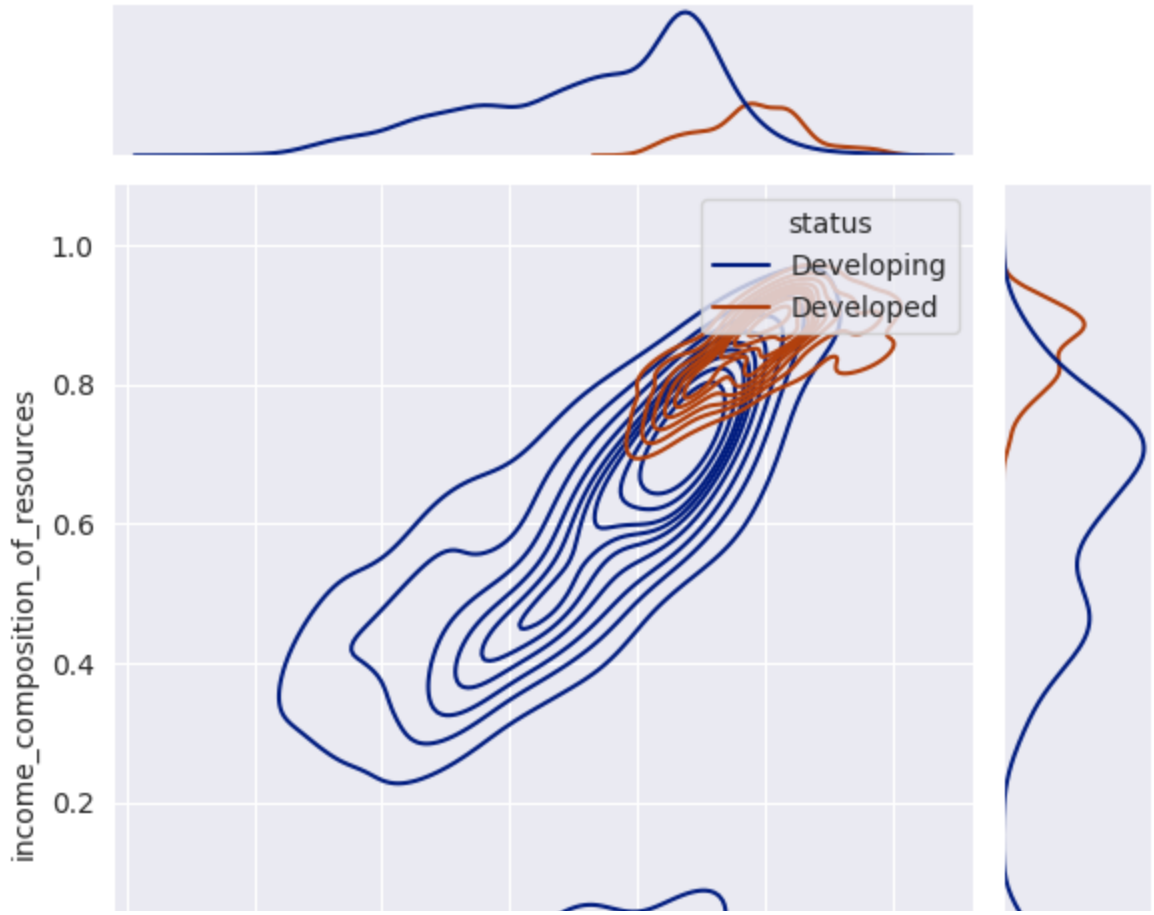
> As the year of schooling increases it is observed that the life expectancy also increases.

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)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
```

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.

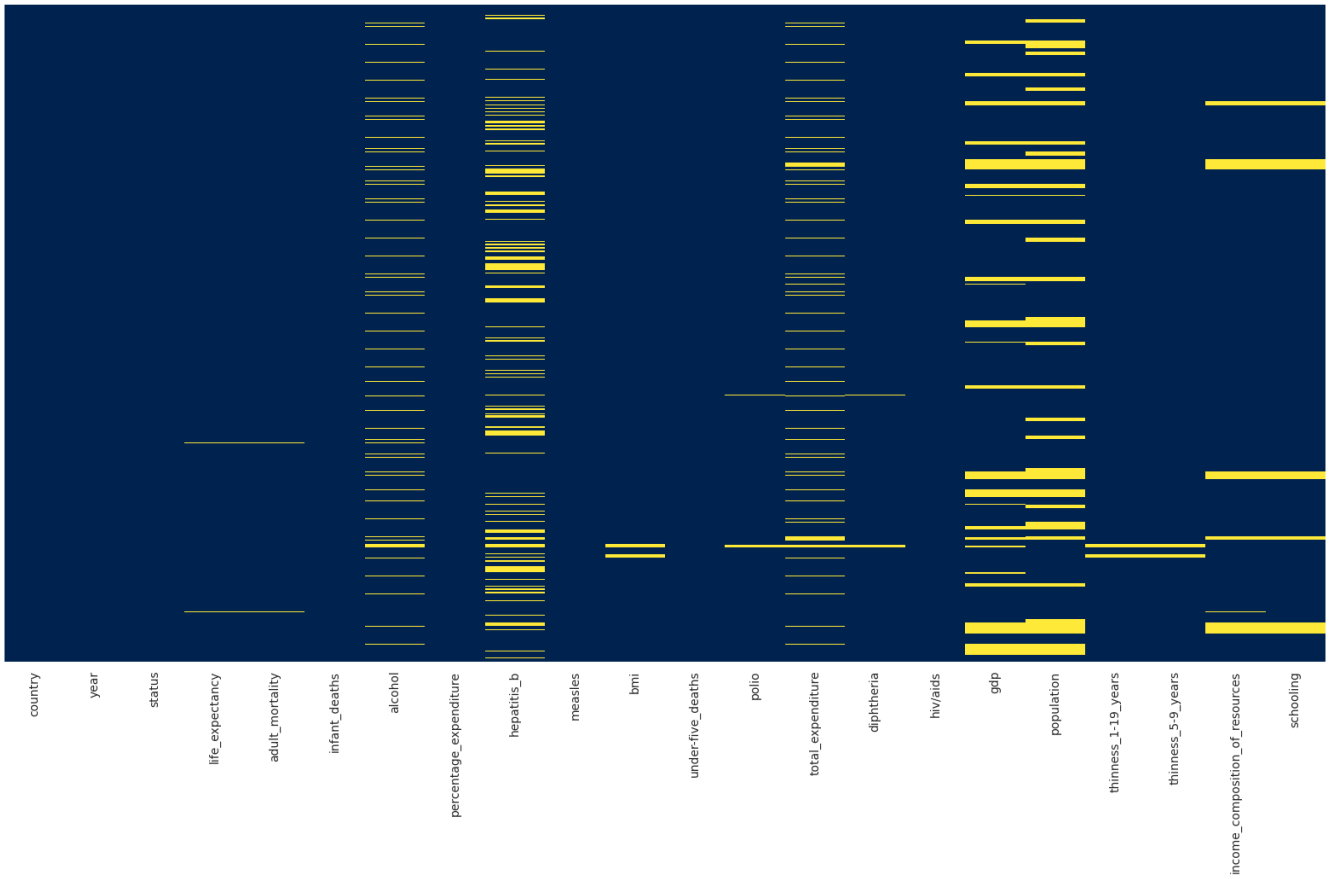
CHECK THE NULL VALUES

```
#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
```

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

```
<Axes: >
```




```
#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
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
```

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.

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
```

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.

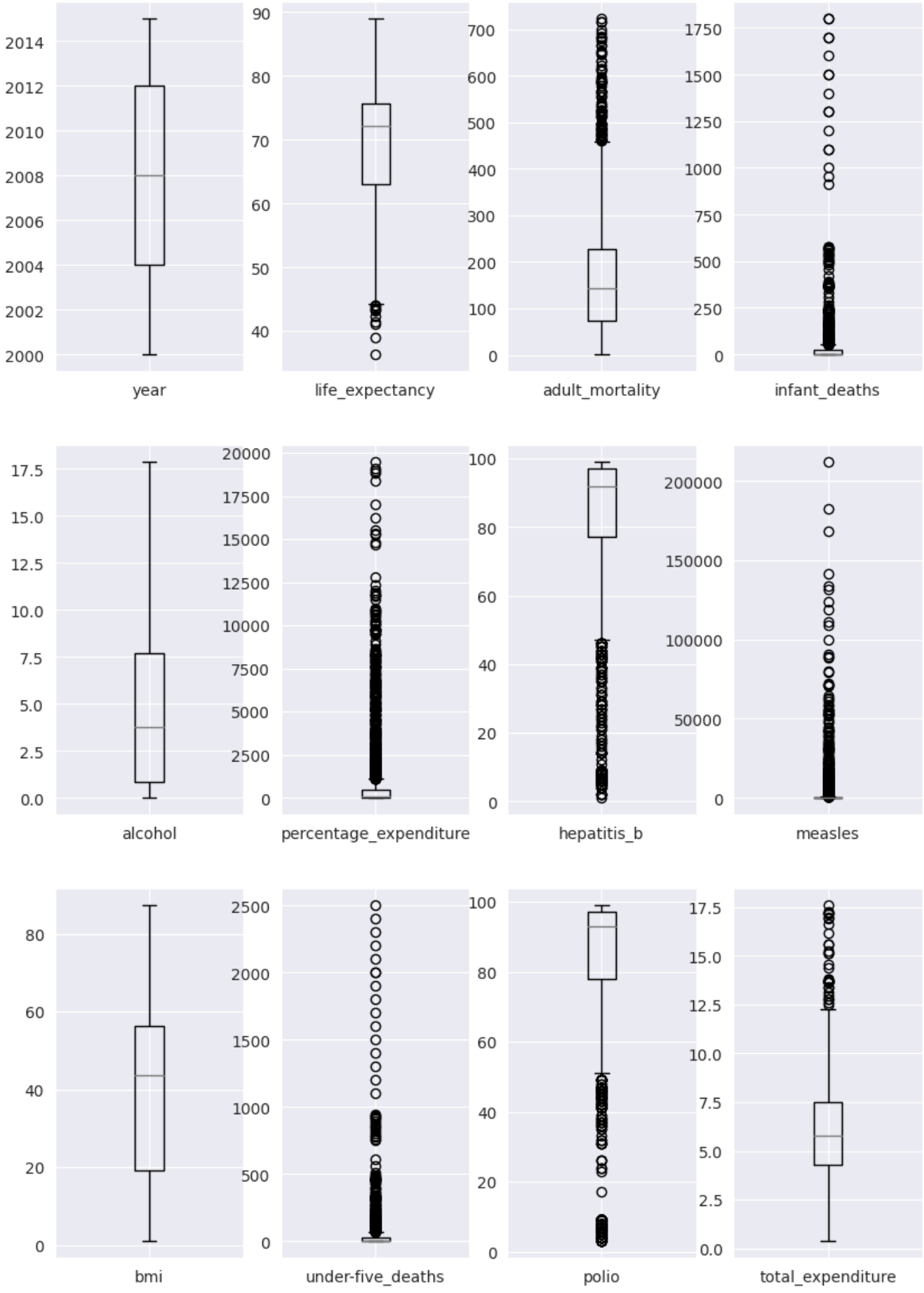
> There are various techniques used for data imputation, including:

> 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

```
#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)
```

```
<ipython-input-463-71470ecf1252>:5: FutureWarning:

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	42.8	614.0	44

ignoring 'palette' because no 'hue' variable has been assigned.

2938 rows × 22 columns

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

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

```
#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/

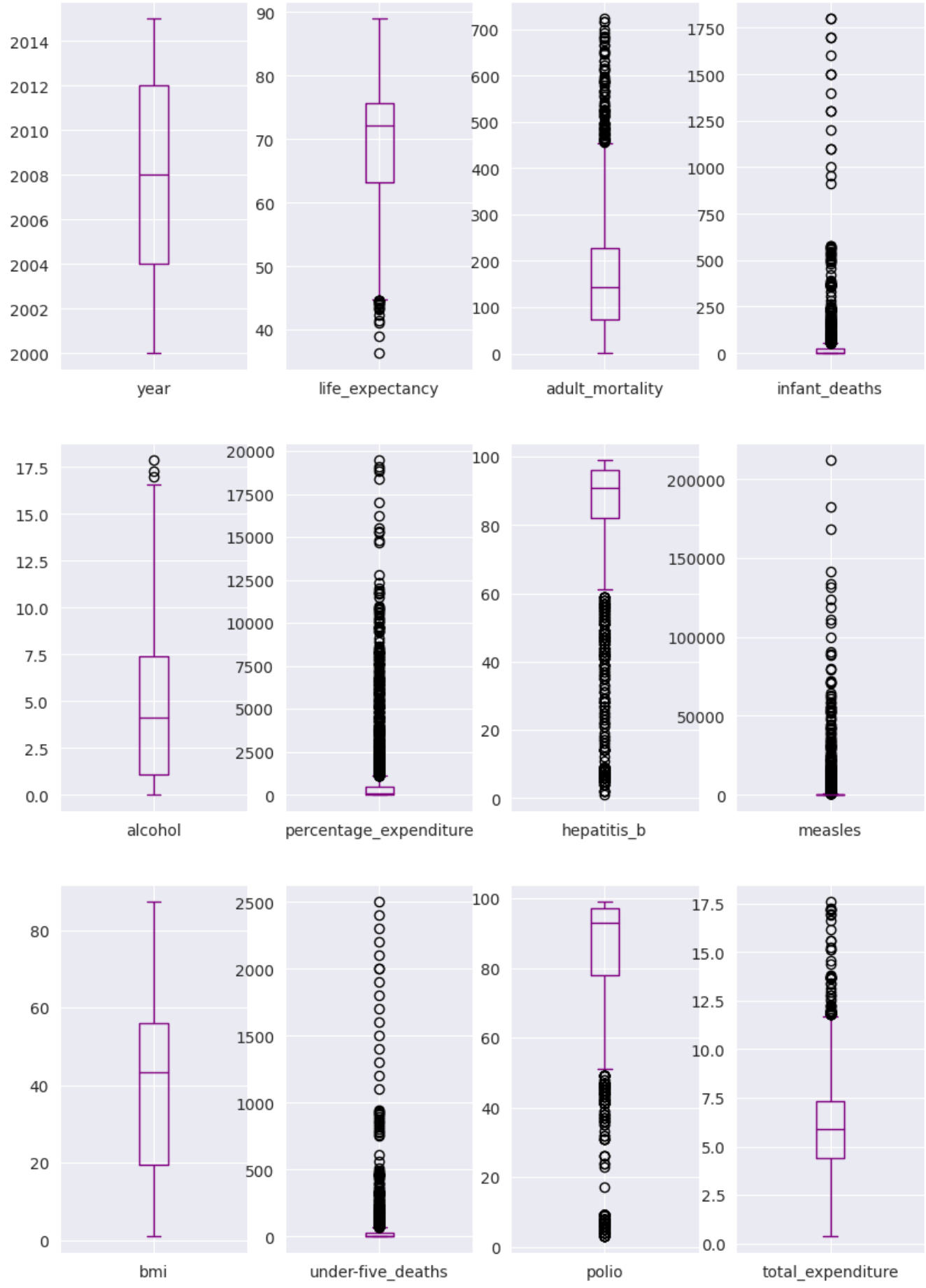
https://www.pluralsight.com/guides/cleaning-up-data-from-outliers

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

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

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

```
#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.

---

25 25

```
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
/usr/local/lib/python3.9/dist-packages/pandas/core/arraylike.py:402: RuntimeWarn

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/

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
```

```
#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

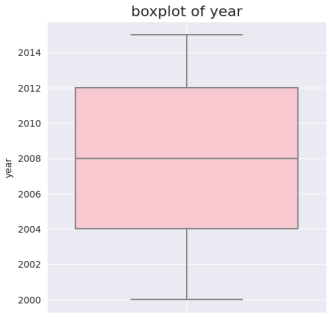
```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
```

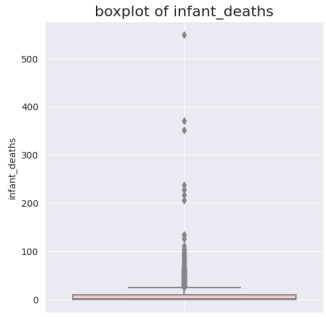
```
columns = list(df_outlier_original.columns)
rem_columns = ['life_expectancy',
 'adult_mortality',
 'alcohol',
 'percentage_expenditure',
 'hepatitis_b',
 'polio',
 'total_expenditure',
 'diphtheria',
 'hiv/aids',
 'gdp',
 'population',
 'thinness_1-19_years',
 'thinness_5-9_years',
 'income_composition_of_resources',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

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

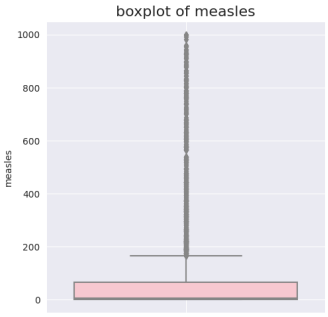
```
19
['year',
 'infant_deaths',
 'measles',
 'under-five_deaths',
 'log_percentage_expenditure',
 'log_population',
 'log_gdp',
 'winz_life_expectancy',
 'winz_adult_mortality',
 'winz_alcohol',
 'winz_hepatitis_b',
 'winz_polio',
 'winz_total_expenditure',
 'winz_diphtheria',
 'winz_income_composition_of_resources',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

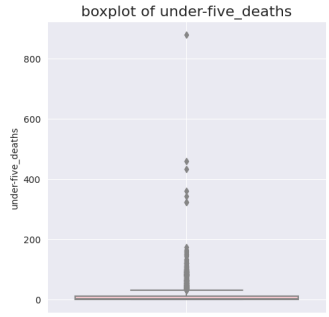
```
plt.figure(figsize=(25,40))
#sns.boxplot(data=data_norm)
i=0
```

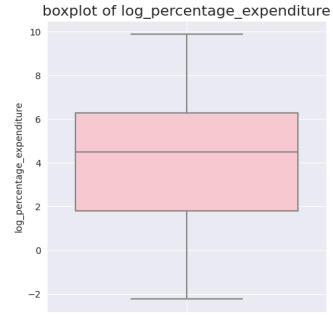
```
for column in required_columns:
  # i+=1
  # plt.subplot(7,3,i)
  # #sns.displot(df_combined_new[column], kde=True, bins=60, color='purple')
  # plt.hist(df_outlier_original[column],color="purple")
  # plt.title(f'Histogram of {column}', fontsize=14)
  # plt.grid(True)

  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()
```

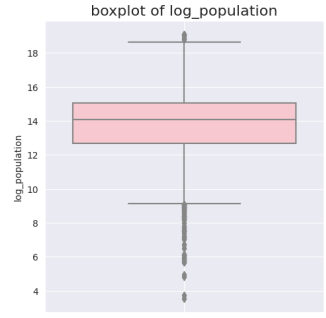

boxplot of year
year

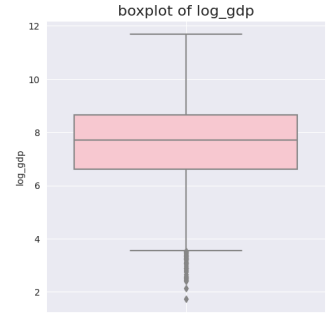

boxplot of infant_deaths
infant_deaths

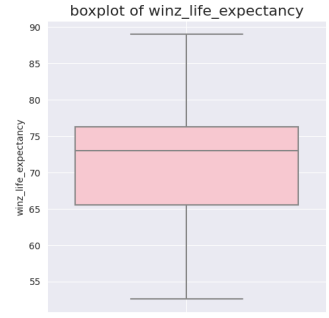

boxplot of measles
measles


boxplot of under-five_deaths
under-five_deaths

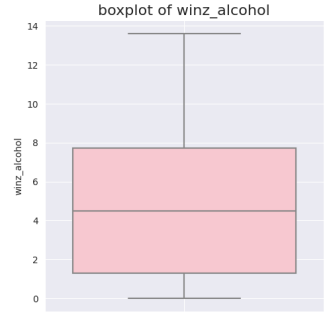

boxplot of log_percentage_expenditure
log_percentage_expenditure

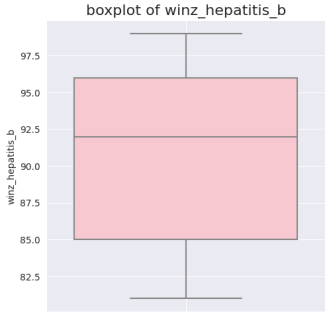

boxplot of log_population
log_population

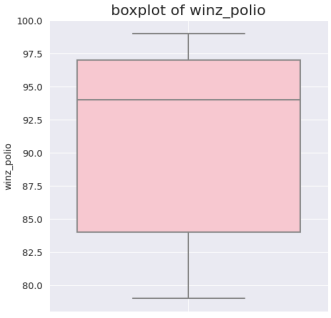

boxplot of log_gdp
log_gdp

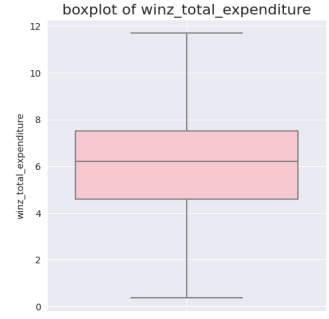

boxplot of winz_life_expectancy
winz_life_expectancy

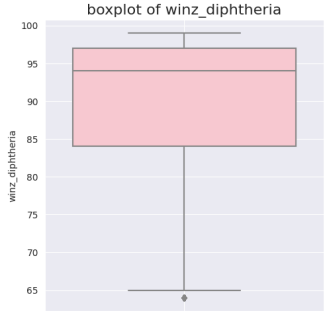

boxplot of winz_adult_mortality
winz_adult_mortality

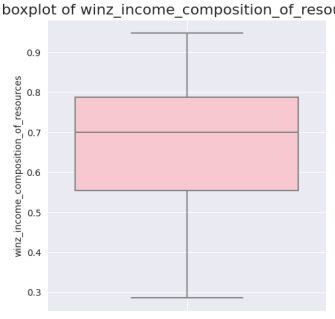

boxplot of winz_alcohol
winz_alcohol

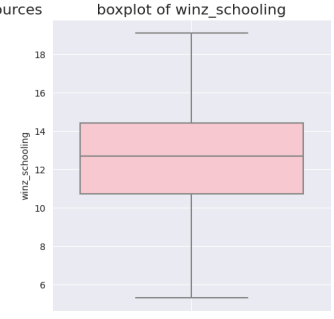

boxplot of winz_hepatitis_b
winz_hepatitis_b

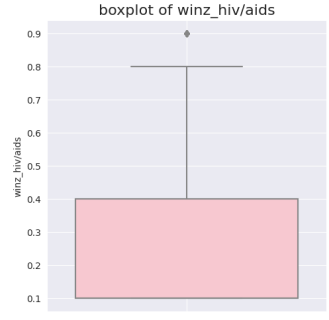

boxplot of winz_polio
winz_polio

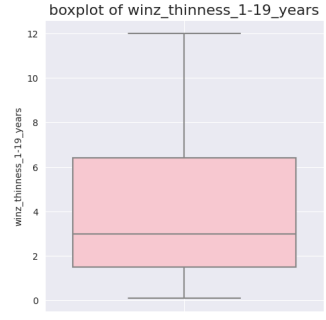

boxplot of winz_total_expenditure
winz_total_expenditure

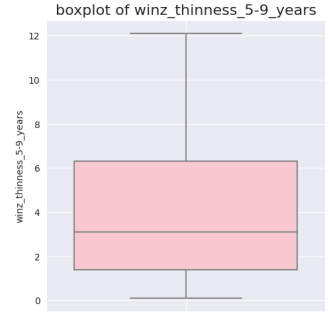

boxplot of winz_diphtheria
winz_diphtheria


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

```
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

```
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 and data preprocessing. 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.

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1069: UserWarning:
```

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.

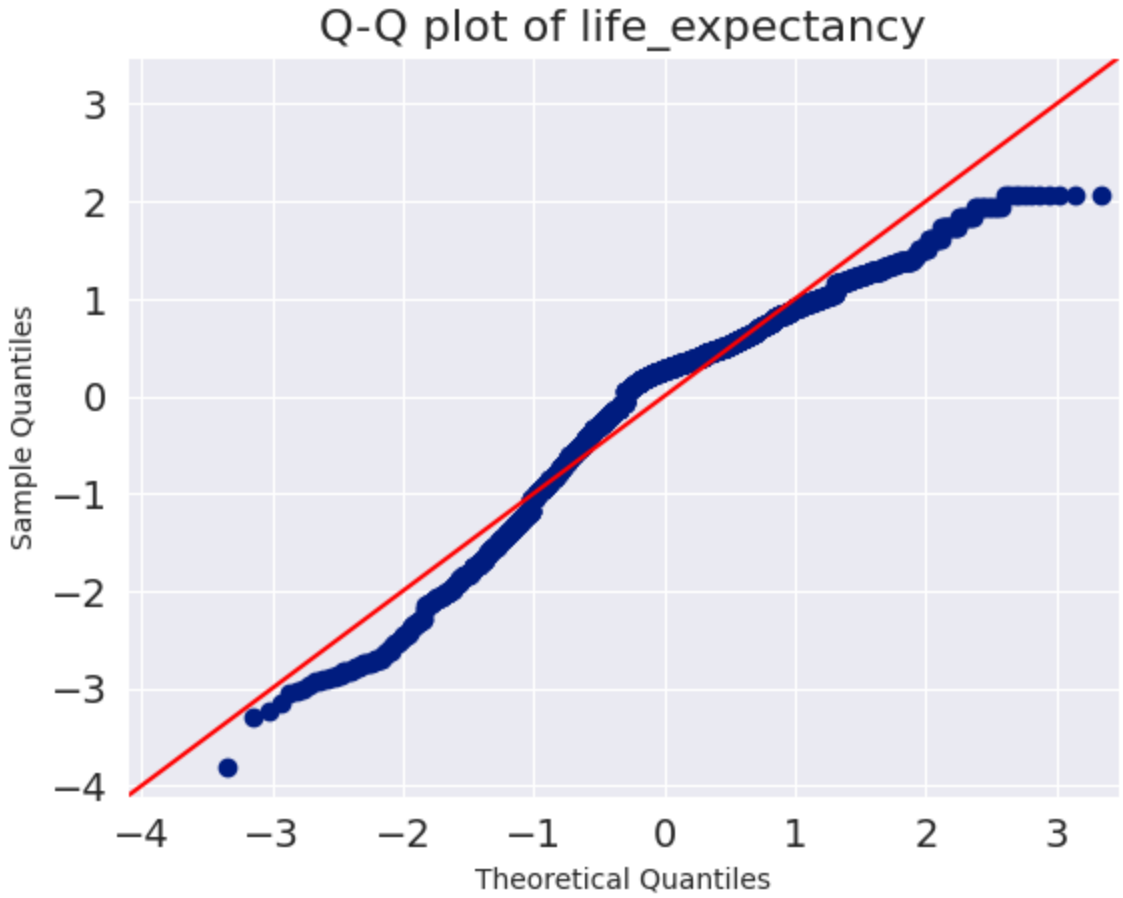
Q-Q PLOT

> 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.

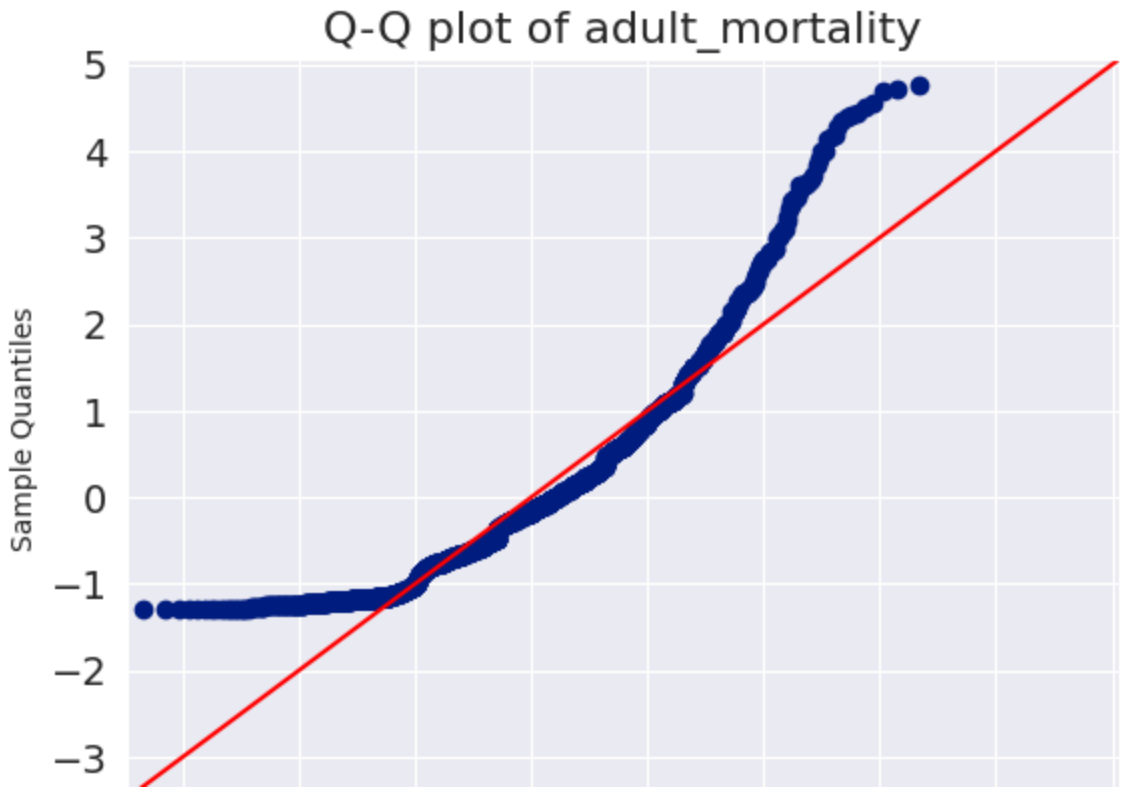
```
#What are the likely distributions of the numeric variables?
#checking the distribution of independent variables
#QQ-PLOT
```

```
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

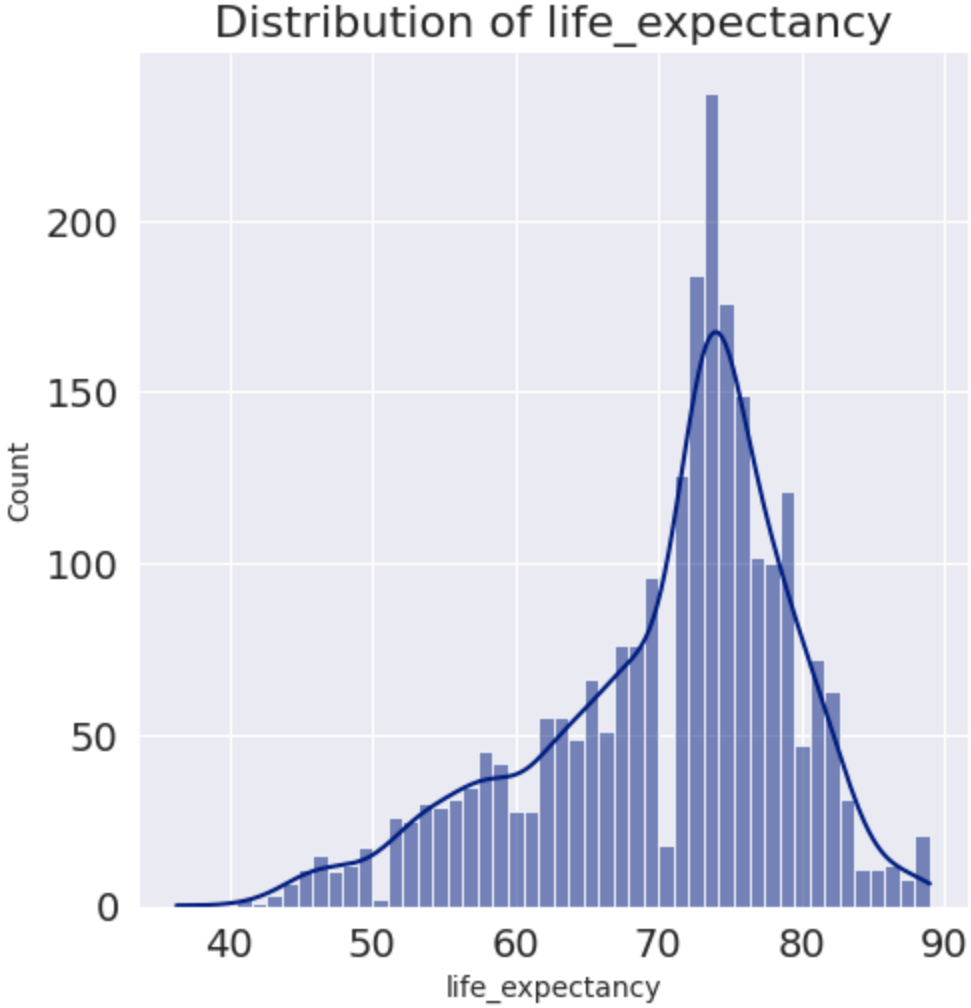
- From the above Q-Q Plot, we can see that almost all the independent variables/features are roughly following normal distribution
- There are few outliers which can be further scaled by using one of the following methods
- StandardScaler
- MinMaxScaler
- RobustScaler
- Normalizer
- We will be using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

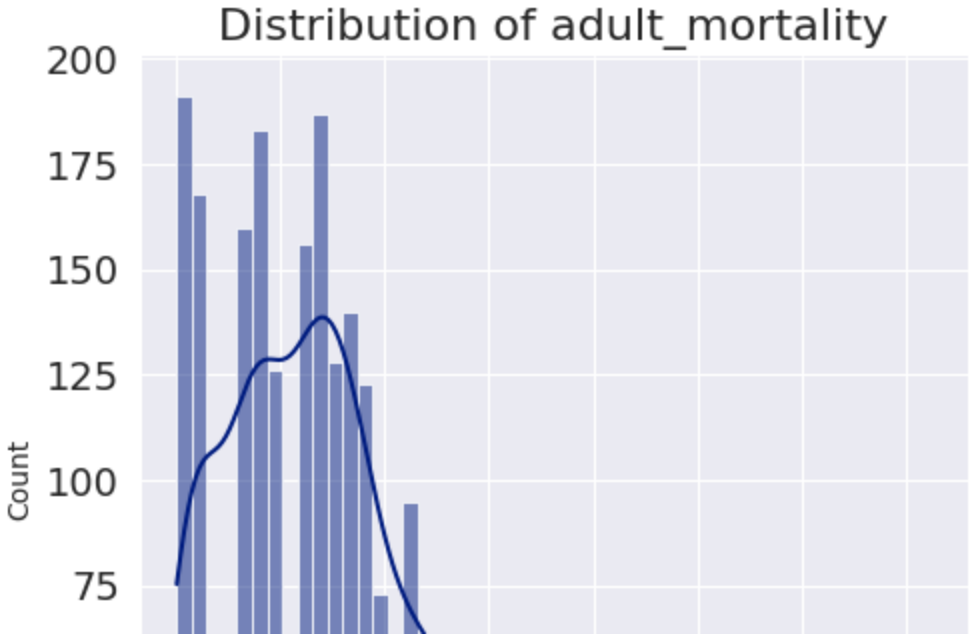
HISTOGRAM PLOT

> 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)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:447: RuntimeWarning:

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()
```

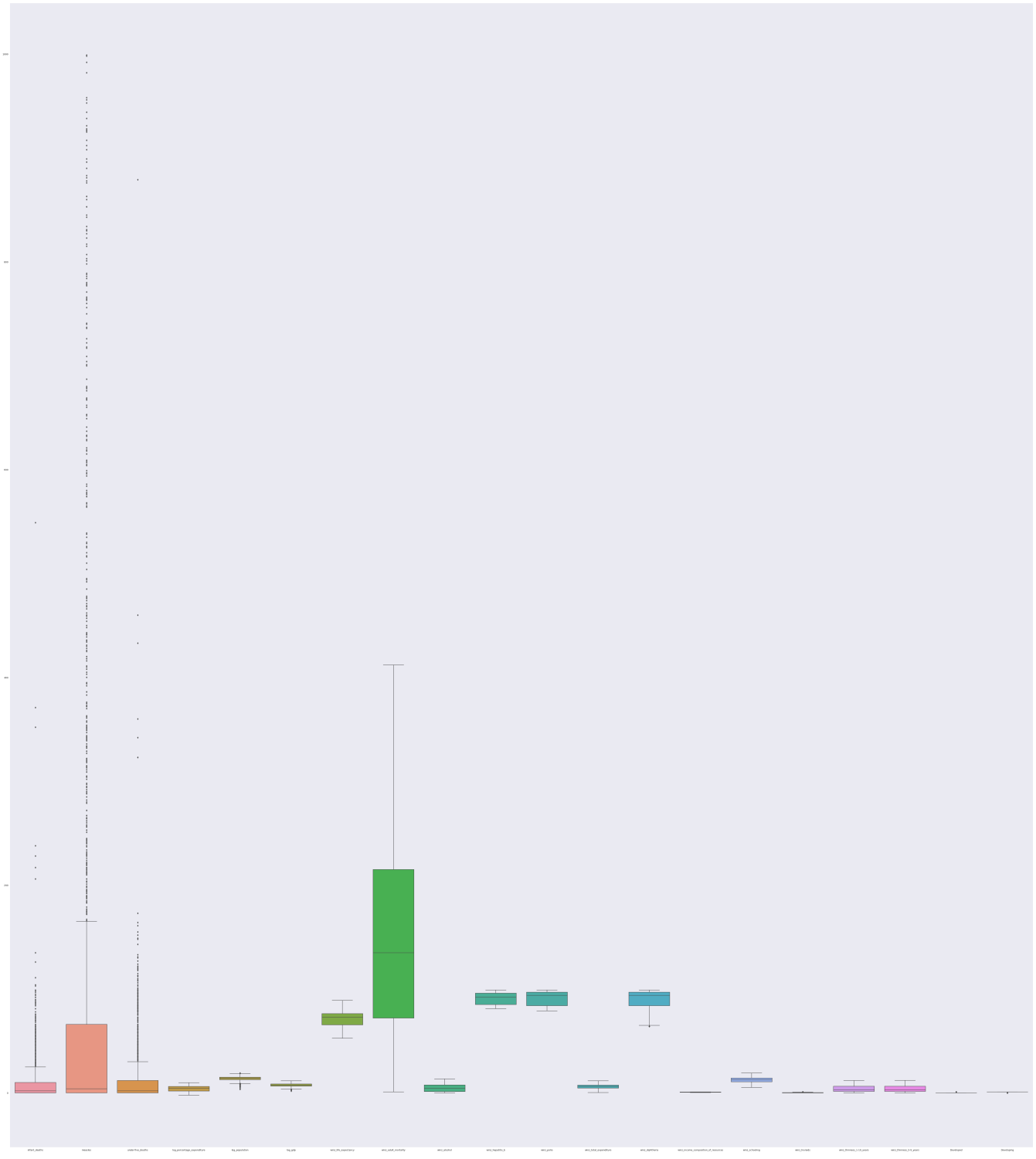
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2413 entries, 16 to 2905
```

```
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

total_expenditure

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	

Visualizing Data Distribution after normalization

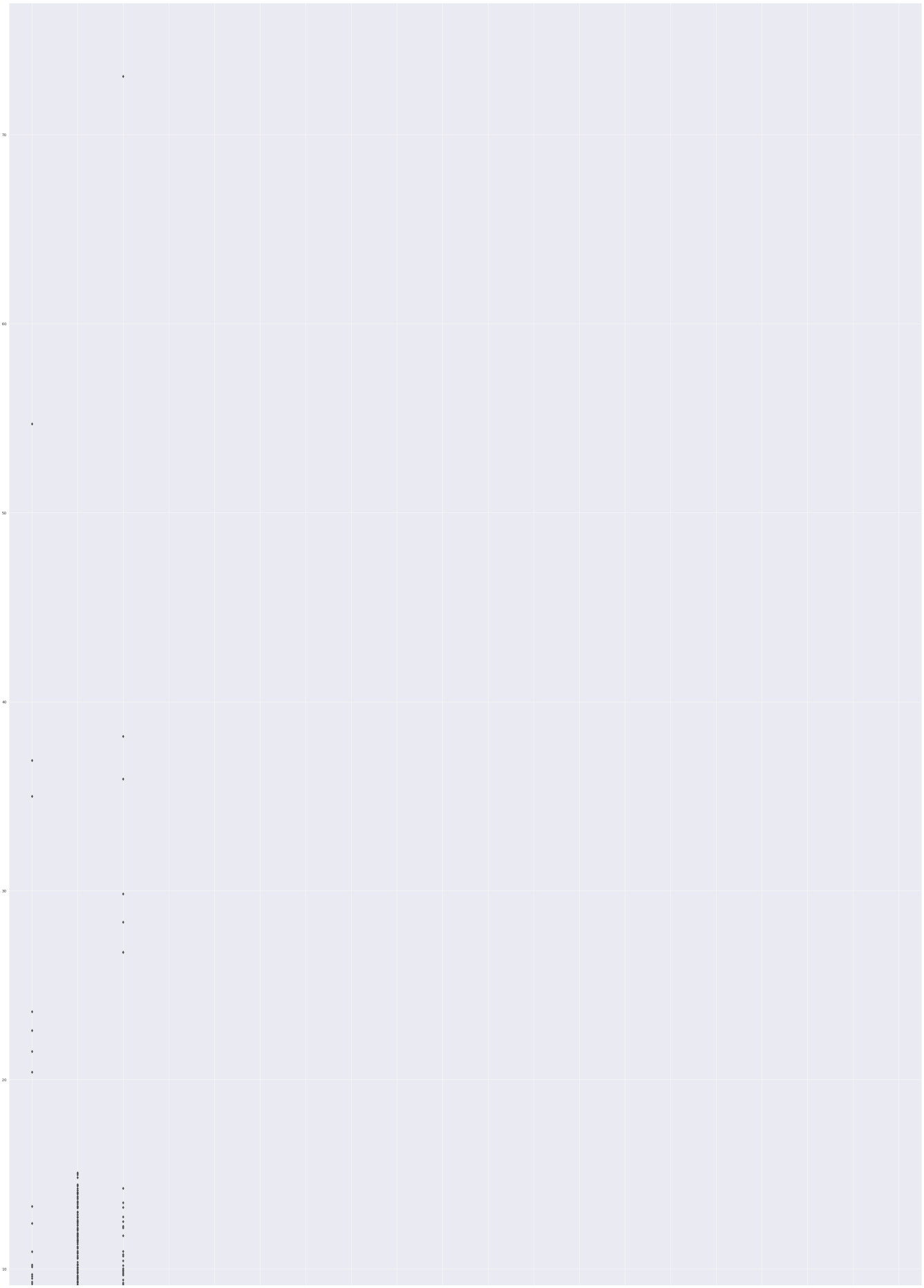
```
plt.figure(figsize=(25,40))
#sns.boxplot(data=data_norm)
i=0

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")
```

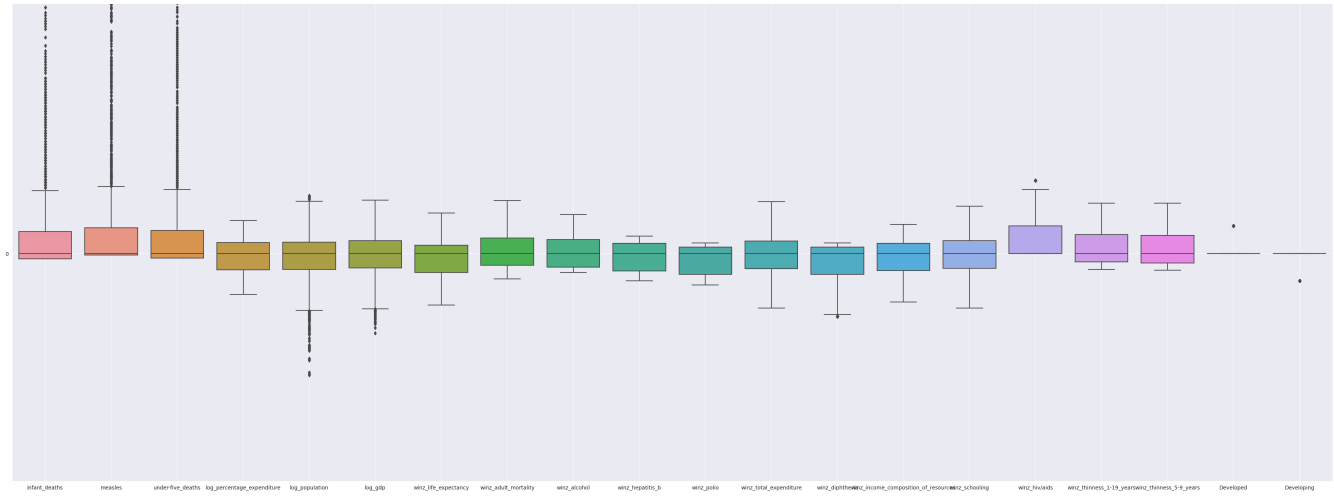
```
  plt.title(f'boxplot of {column}', fontsize=16)
  plt.grid(True)

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

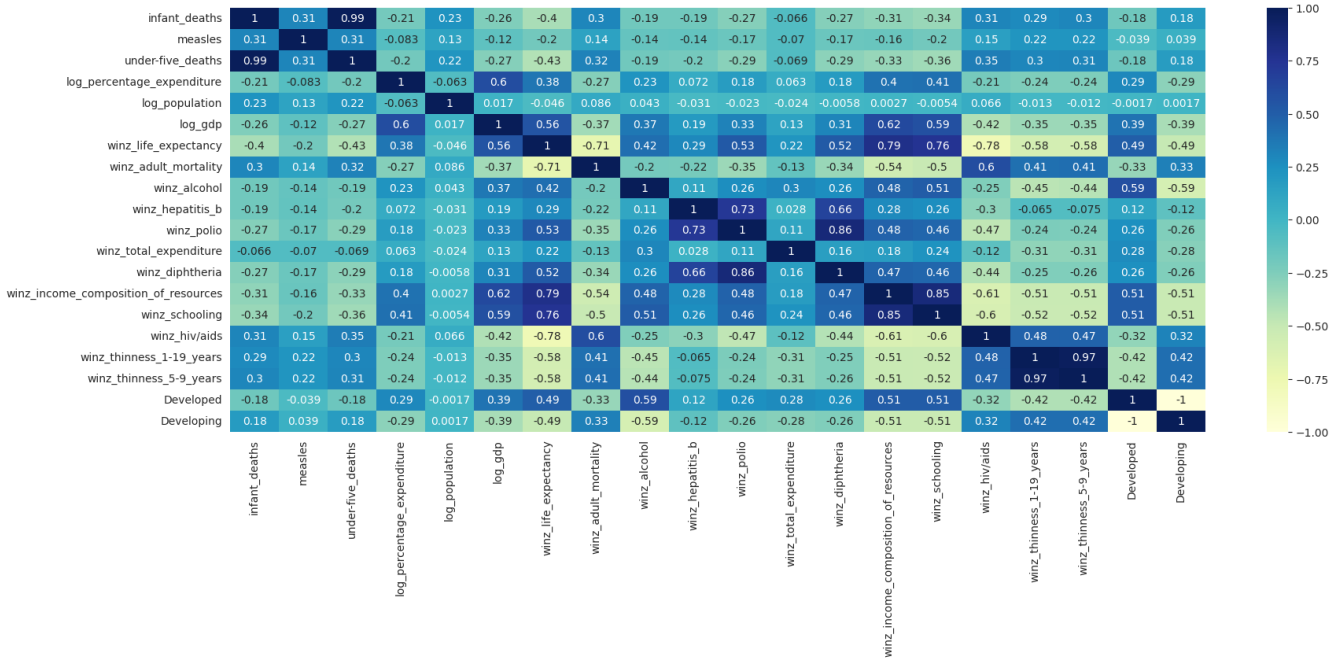


log_gdp

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: >
```

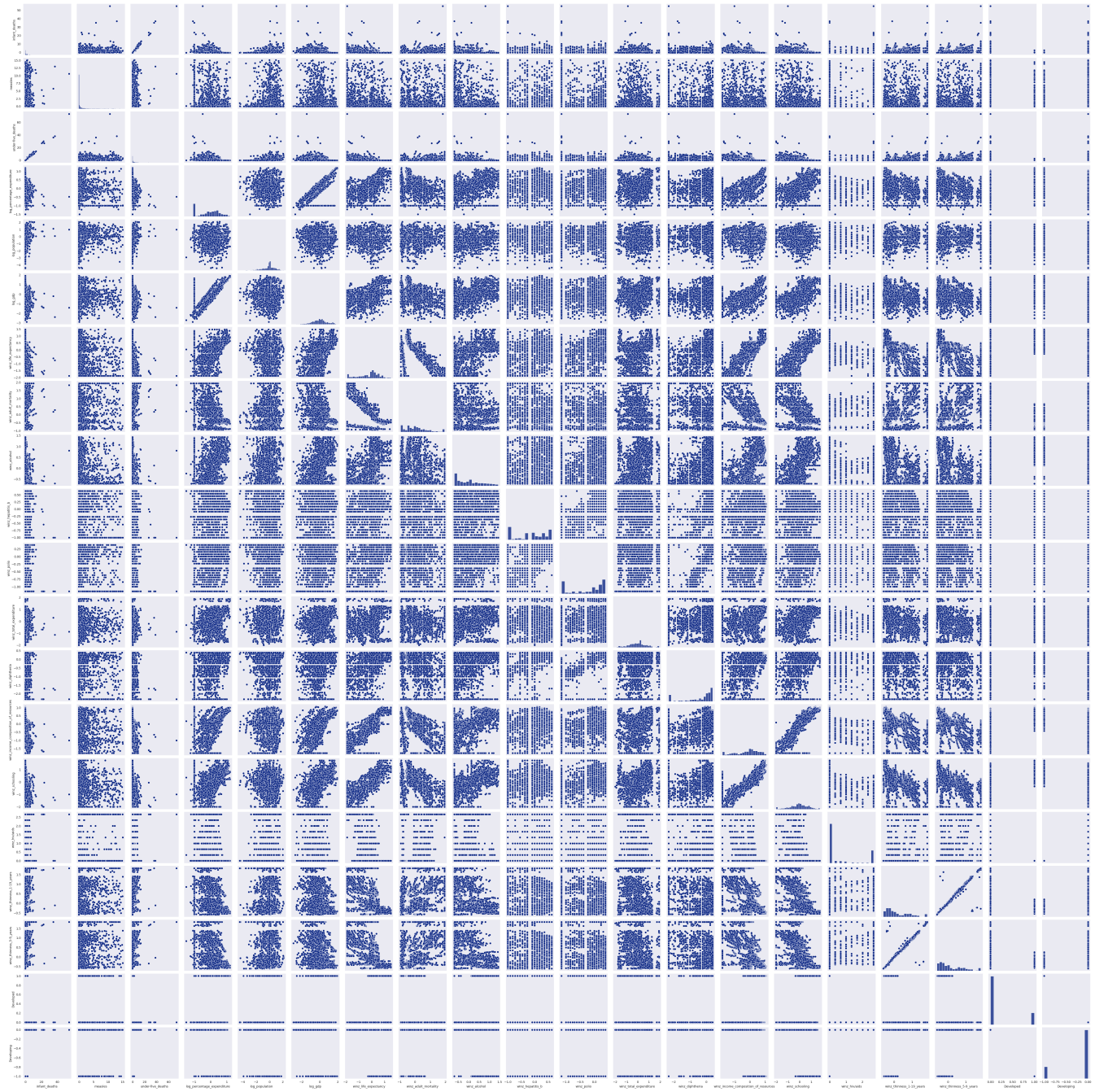


60

Visualizing Pair plot of the correlation between features after normalization

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

```
<seaborn.axisgrid.PairGrid at 0x7f06add734c0>
```



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.

```
#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

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.

```
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
4	5.058668	2.459297e-02
5	1105.002155	8.674719e-200
6	2390.088506	0.000000e+00
7	502.436693	3.086049e-101

BUILDING THE MODEL USING LINEAR REGRESSION

10 121.556951 1.302518e-27

TRAIN, TEST SPLIT

Developing

```
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
```

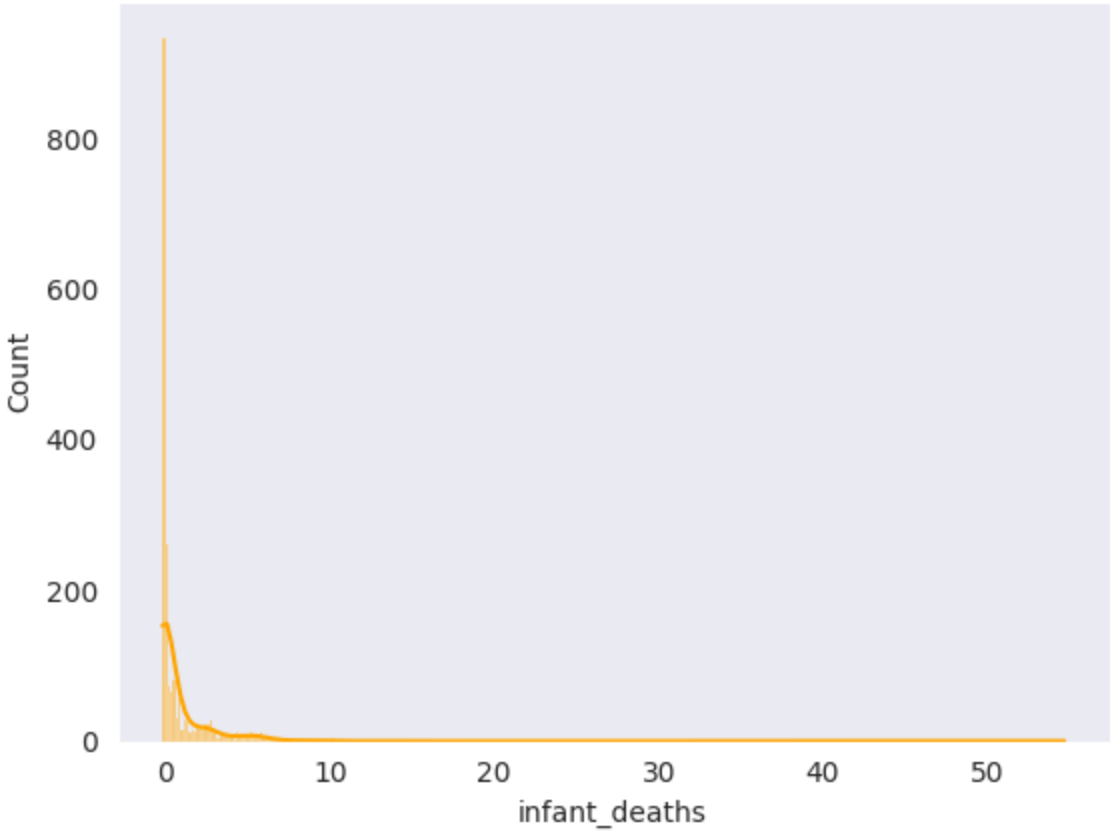
```
 4   winz_life_expectancy                  2413 non-null   float64
 5   winz_adult_mortality                  2413 non-null   float64
 6   winz_hepatitis_b                      2413 non-null   float64
 7   winz_polio                            2413 non-null   float64
 8   winz_total_expenditure                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
```

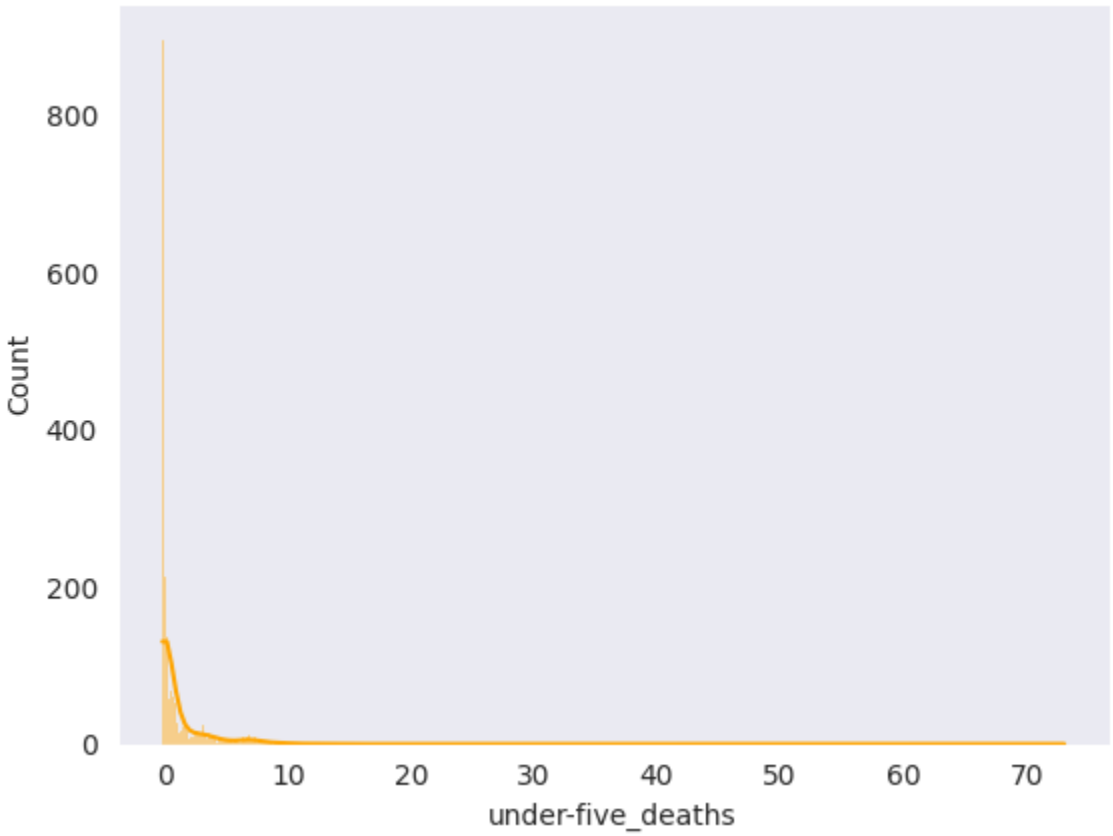
```
X_train.head()
```

	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")
```

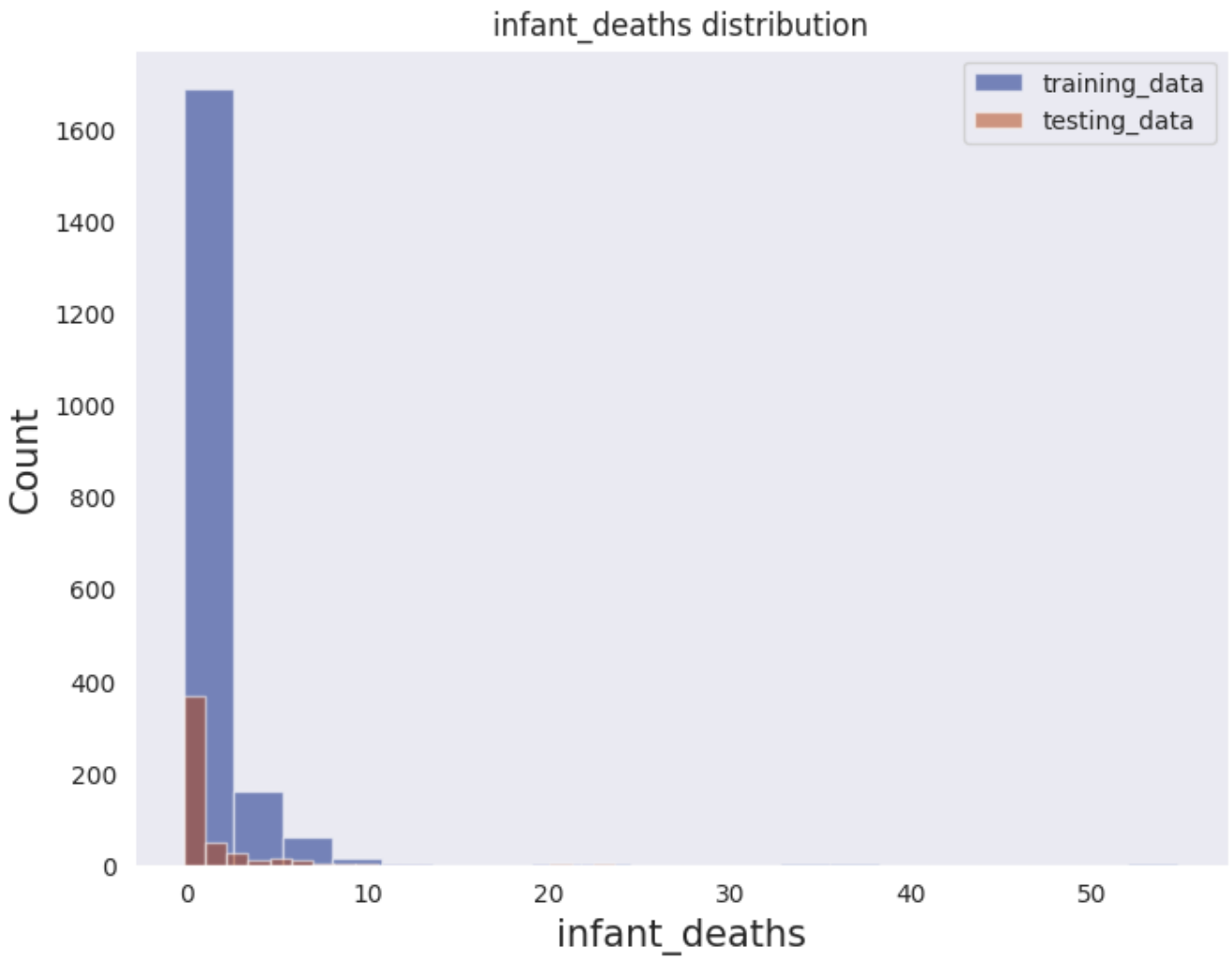

800
600
400
200
0
Count
0
10
20
30
40
50
infant_deaths


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
1600
1400
1200
1000
800
600
400
200
0
0
10
20
30
40
50
infant_deaths


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

INFERENCE

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

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

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

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

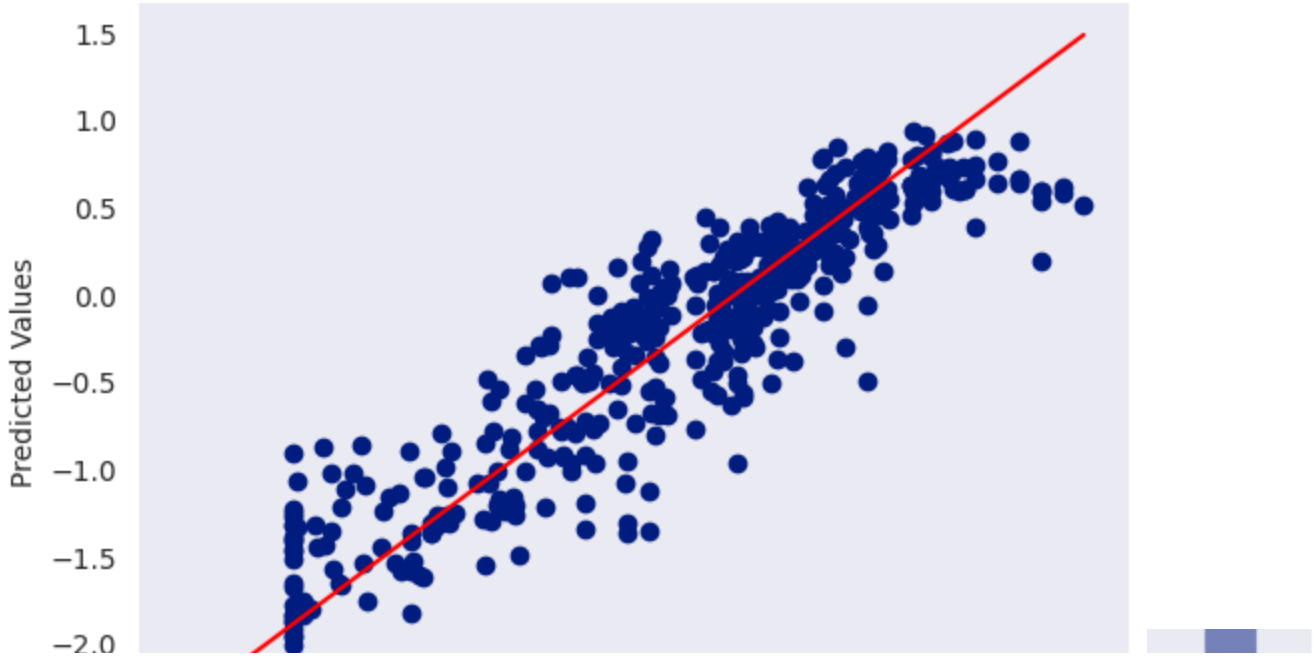
```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

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 0x7f0633942730>]
```



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

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.

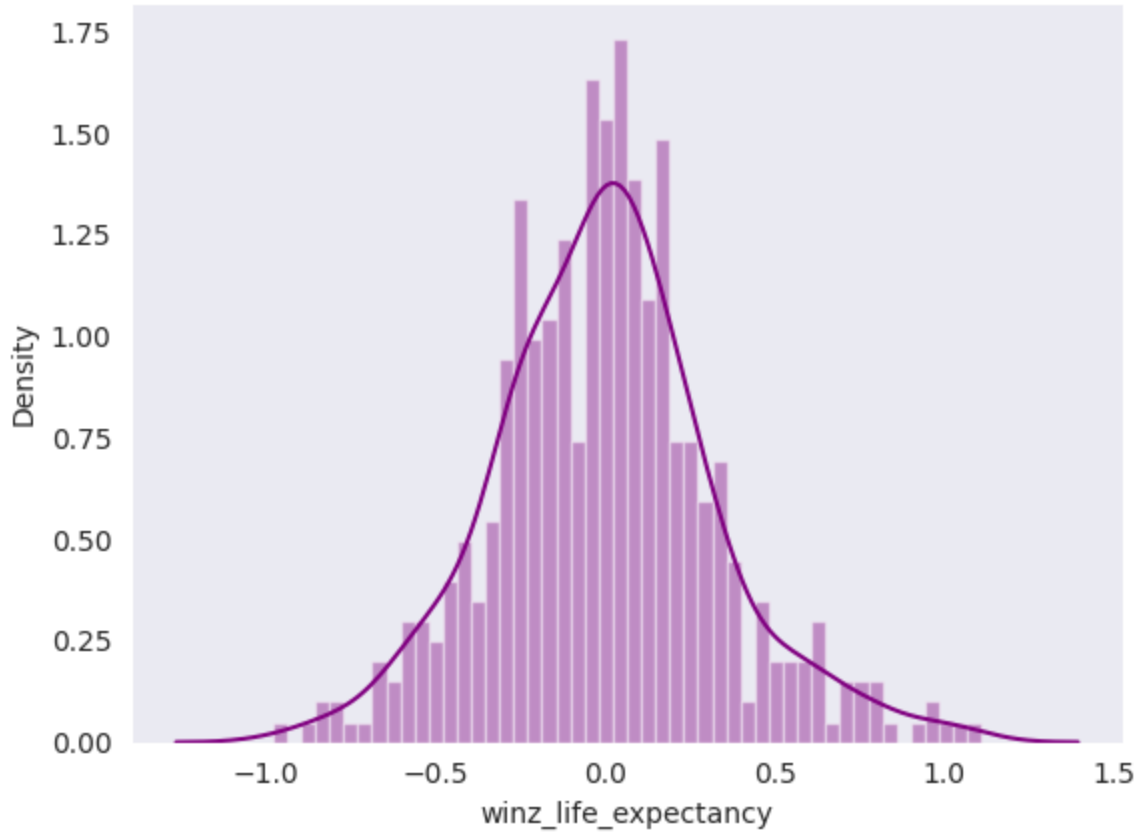
> https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/regression-library/a/introduction-to-residuals

---

winz income composition of resources distribution

```
#RESIDUALS

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

```
<ipython-input-500-b8771af6b358>:3: UserWarning:


`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

600

CALCULATING MSE AND VARIANCE USING USER-DEFINED FUNCTION AND COMPARING THE VALUES WITH 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/

FUNCTION DEFINED TO IMPLEMENT MSE AND VARIANCE MATHEMATICALLY AND COMPARE IT WITH THE VALUE OBTAINED USING SCIKIT BUILT-IN FUNCTION

```
def statistical_measures(y_original, y_predictions):

  length = len(y_original)

  residuals = y_original-y_predictions
  residual_mean = residuals.mean()


  print("-------------------------------------------------------------------------------")
  print("\nMEAN SQUEARE ERROR (MSE)")
  print("-------------------------------------------------------------------------------")
  print("MATHEMATICAL FORMULA FOR FINDING MSE:")
  print('''
╔═══════════════════════════════════════════════╗
║  MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2          ║
╚═══════════════════════════════════════════════╝
''')


  #FINDING MSE USING THE MATHEMATICAL FORMULA
  mse_formula = round((pow((residuals),2)).sum()/length,2)
  print("MSE VALUE USING MATHEMATICAL FORMULA --> ", mse_formula )

  #FINDING MSE USING THE SCIKIT INBUILT FORMULA
  print('MSE VALUE USING SCIKIT FUNCTION: ', round(metrics.mean_squared_error(y_origi

  print("-------------------------------------------------------------------------------")
  print("-------------------------------------------------------------------------------")
```

```
  print("\n RESIDUAL VARIANCE")
  print("--------------------------------------------------------------------------------
··print("MATHEMATICAL·FORMULA·FOR·FINDING·RESIDUAL·VARIANCE:")
··print('''
╔═══════════════════════════════════════════════════════════╗
╠··Residual·Variance·=·1/(n-k-1)·*·∑(i=1·to·n)·(yi·-·ŷi)^2··╠
╚═══════════════════════════════════════════════════════════╝
''')


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

  #FINDING VARIANCE USING THE MATHEMATICAL FORMULA
  var_formula = round((pow((residuals - residual_mean),2)).sum()/length,5)
  print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )

  #FINDING VARIANCE USING THE SCIKIT INBUILT FORMULA
  var = round(statistics.variance(list(residuals)),5)
  print("VARIANCE USING SCIKIT FUNCTION --> ", var)
  print("--------------------------------------------------------------------------------
```

FUNCTION DEFINED TO PLOT THE RESIDUAL

```
def plot_residual(y_original,y_predictions):


  residual =  y_original - y_predictions

  # Importing the required libraries
  import matplotlib.pyplot as plt
  import seaborn as sns

  # SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
  print("SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES")
  fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4))
  sns.scatterplot(x=y_original, y=y_predictions, ax=ax1)
  ax1.plot([y_original.min(), y_original.max()], [y_original.min(), y_original.max()]
  ax1.set_xlabel('True Values')
  ax1.set_ylabel('Predicted Values')

  # RESIDUAL PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
  sns.distplot((residual), bins=50, color='orange', ax=ax2)
  ax2.set_xlabel('Residual')
  ax2.set_ylabel('Frequency')

  # Display the plots
```

```
  plt.tight_layout()
  plt.show()
```

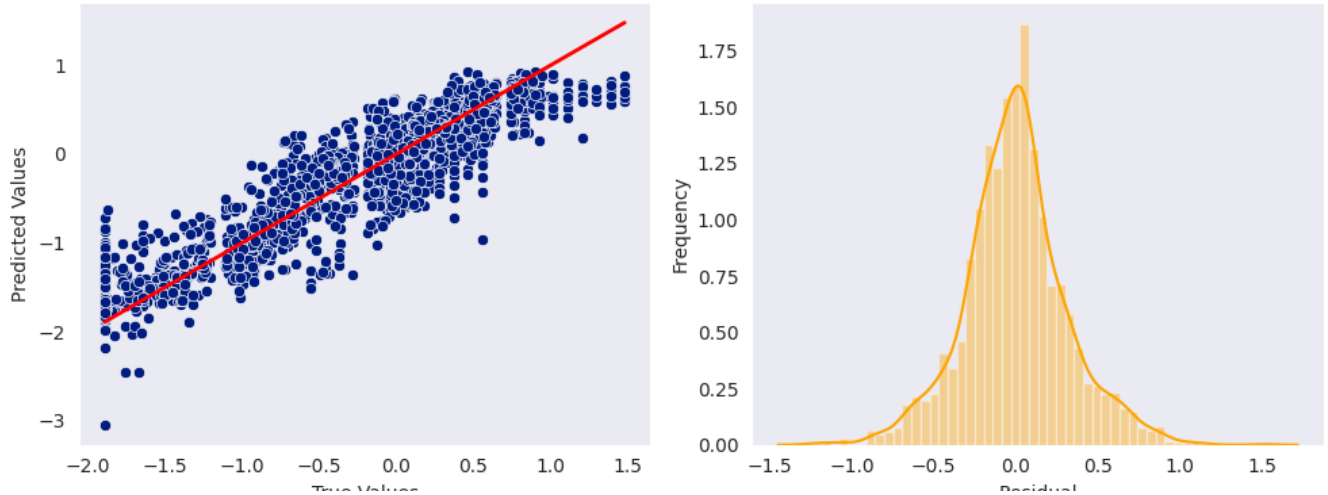
Implementation on TRAINING DATA

```
y_predictions = lm.predict(X_train)
statistical_measures(y_train, y_predictions)
```

```
    ------------------------------------------------------------------------------

    MEAN SQUEARE ERROR (MSE)
    ------------------------------------------------------------------------------
    MATHEMATICAL FORMULA FOR FINDING MSE:

    ╔═══════════════════════════════════════════════╗
    ║  MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2          ║
    ╚═══════════════════════════════════════════════╝

    MSE VALUE USING MATHEMATICAL FORMULA -->  0.1
    MSE VALUE USING SCIKIT FUNCTION:  0.1
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------

     RESIDUAL VARIANCE
    ------------------------------------------------------------------------------
    MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:

    ╔═══════════════════════════════════════════════════════╗
    ║  Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2  ║
    ╚═══════════════════════════════════════════════════════╝

    Mean value of the residuals -->  1.2057137096447296e-16

    VARIANCE USING MATHEMATICAL FORMULA -->  0.09798
    VARIANCE USING SCIKIT FUNCTION -->  0.09803
    ------------------------------------------------------------------------------
```

```
plot_residual(y_train,y_predictions)
```

```
SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
<ipython-input-503-38dca7de071a>:19: UserWarning:


`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
```



Implementation on TEST DATA

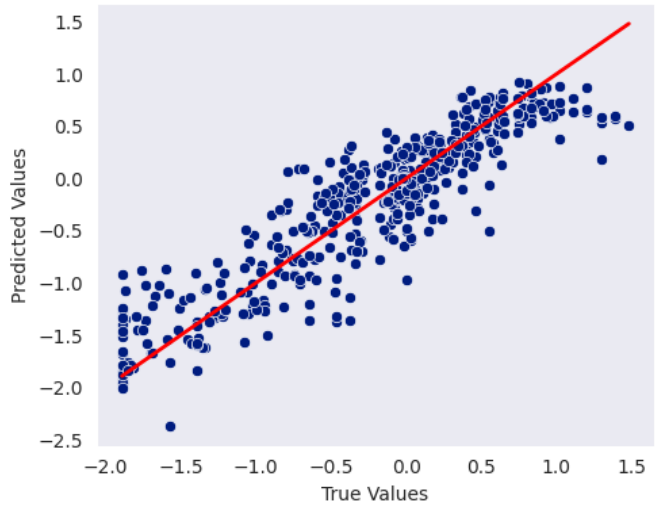
```
y_predictions = lm.predict(X_test)
statistical_measures(y_test, y_predictions)
```

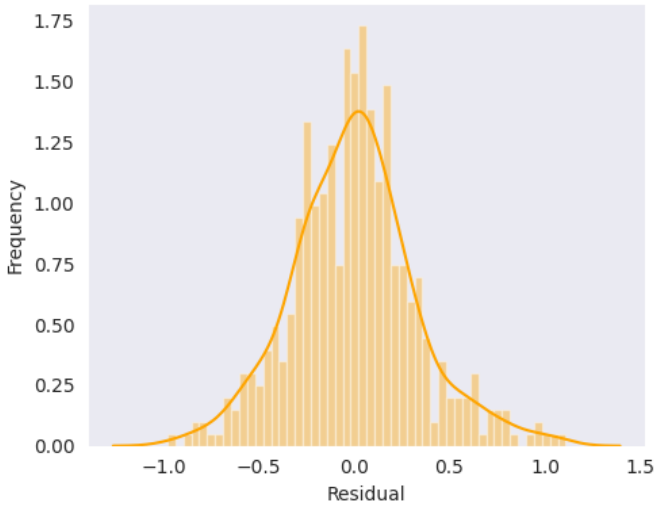
```
--------------------------------------------------------------------------------

MEAN SQUEARE ERROR (MSE)
--------------------------------------------------------------------------------
MATHEMATICAL FORMULA FOR FINDING MSE:

  MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2

MSE VALUE USING MATHEMATICAL FORMULA -->  0.11
MSE VALUE USING SCIKIT FUNCTION:  0.11
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

 RESIDUAL VARIANCE
```

```
--------------------------------------------------------------------------------
MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:

╔═══════════════════════════════════════════════════════╗
║  Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2  ║
╚═══════════════════════════════════════════════════════╝

Mean value of the residuals -->  -0.0005757295053002555

VARIANCE USING MATHEMATICAL FORMULA -->  0.10613
VARIANCE USING SCIKIT FUNCTION -->  0.10635
--------------------------------------------------------------------------------
```

```
plot_residual(y_test,y_predictions)
```

```
SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
<ipython-input-503-38dca7de071a>:19: UserWarning:


`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
```





INFERENCE

1. I was able to gain a better understanding of the mathematical implementation of prediction metrics by comparing the results of my self-defined function and the scikit built-in function for calculating Mean Squared Error (MSE) and Residual Variance. I found that both functions gave the same result, which helped me to gain more clarity on these metrics and their usage in analyzing prediction results.

2. The Scatter Plot shows a relatively linear relationship between the predicted values and the actual values, with the points clustered around the diagonal line (y=x). The Histogram plot of the residuals are approximately normal and centered around zero, with no noticeable skewness or outliers. These characteristics suggest that the model is making accurate predictions with a low degree of error.

3. Therefore, Both the Scatter Plot and Histogram of the residuals indicate that the predicted values for both the train and test data are in close agreement with the actual values. This suggests that the model is performing well and can be utilized for further analysis.

WORKED EXAMPLE 2: PREDICTING HEALTH INSURANCE PRICE USING LINEAR REGRESSION

> Health insurance is a means of protecting oneself from financial losses due to medical expenses.
>
> It typically covers the cost of medical treatments, hospitalization, and medications.
>
> The cost of health insurance premiums varies based on factors such as age, gender, pre-existing medical conditions, and geographic location.
>
> Linear regression can be used to predict health insurance premiums based on these and other predictor variables.
>
> Accurate prediction of health insurance costs can help individuals and organizations make informed decisions about their healthcare coverage.

ABOUT THE DATASET

The Health Insurance Price Predict dataset is a publicly available dataset on Kaggle that contains information on individuals' demographic and health-related characteristics, along with their annual health insurance premiums. The dataset includes over 13,000 rows and 7 columns, making it a valuable resource for those interested in developing predictive models for health insurance pricing

dataset: https://www.kaggle.com/code/shubhamptrivedi/health-insurance-price-predict-linear-regression/input

DATA DICTIONARY

1. **age:** age of the primary beneficiary (numeric)
2. **sex:** gender of the primary beneficiary (categorical: 'male' or 'female')
3. **bmi:** body mass index (numeric)
4. **children:** number of children covered by health insurance (numeric)
5. **smoker:**smoking status of the primary beneficiary (categorical: 'yes' or 'no')
6. **region:** region where the primary beneficiary resides (categorical: 'northeast', 'southeast', 'southwest', or 'northwest')
7. **charges:** individual medical costs billed by health insurance (numeric)

The goal of this dataset is to predict health insurance premiums (charges) based on demographic and health-related characteristics of individuals. The dataset does not contain any missing or erroneous values.

READ THE DATA

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

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

```
df_med_original.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

CLEANING THE DATA COLUMNS NAMES

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

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

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

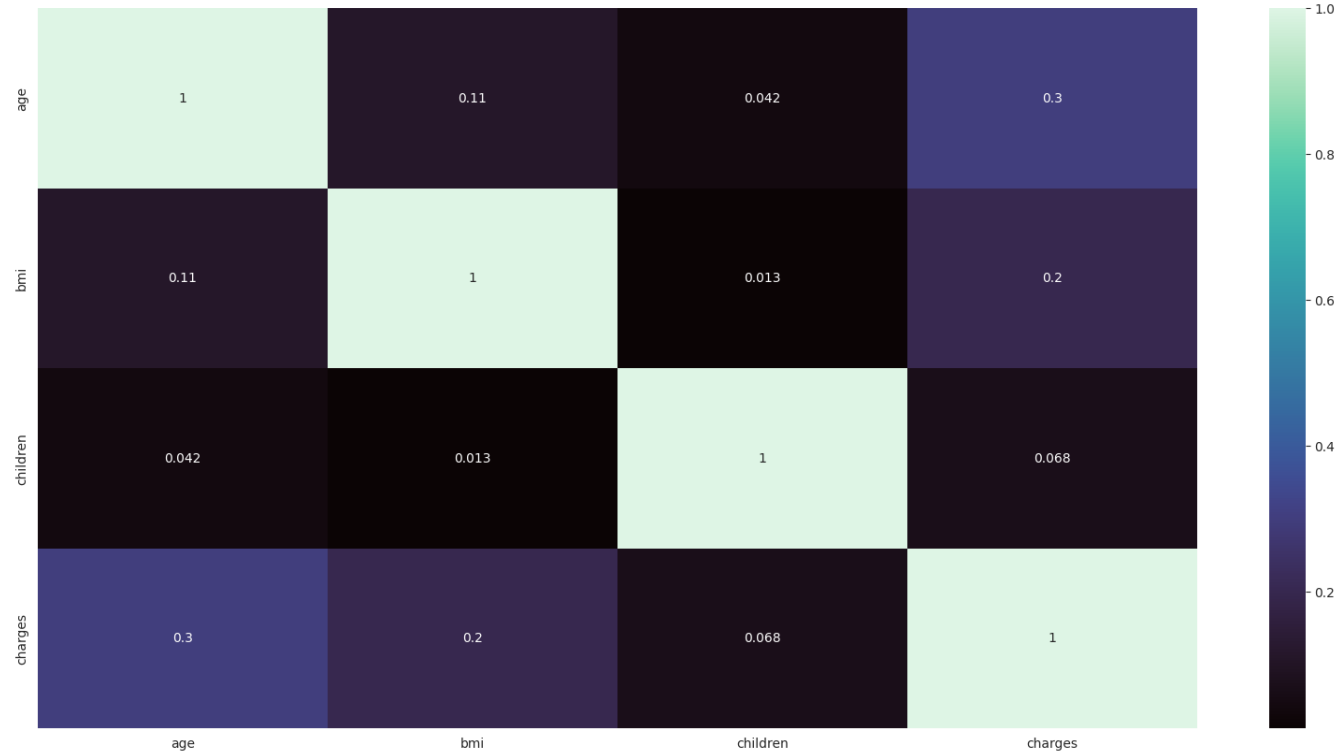
	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

EXPLORATORY DATA ANALYSIS (EDA)

Correlation Heatmap for dependency Visualization

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

```
<ipython-input-513-6570672d3273>:3: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future v

<Axes: >
```



INFERENCE

From this heatmap, we can make several observations:

age, bmi, and charges have the strongest correlations with each other. This is not surprising, as older individuals tend to have higher BMIs and higher healthcare costs.

There is a moderate positive correlation between **age and charges**, indicating that older individuals tend to have higher healthcare costs.

There is a moderate positive correlation between **bmi and charges**, indicating that individuals with higher BMIs tend to have higher healthcare costs.

children has a weak positive correlation with charges, indicating that individuals with more children tend to have slightly higher healthcare costs.

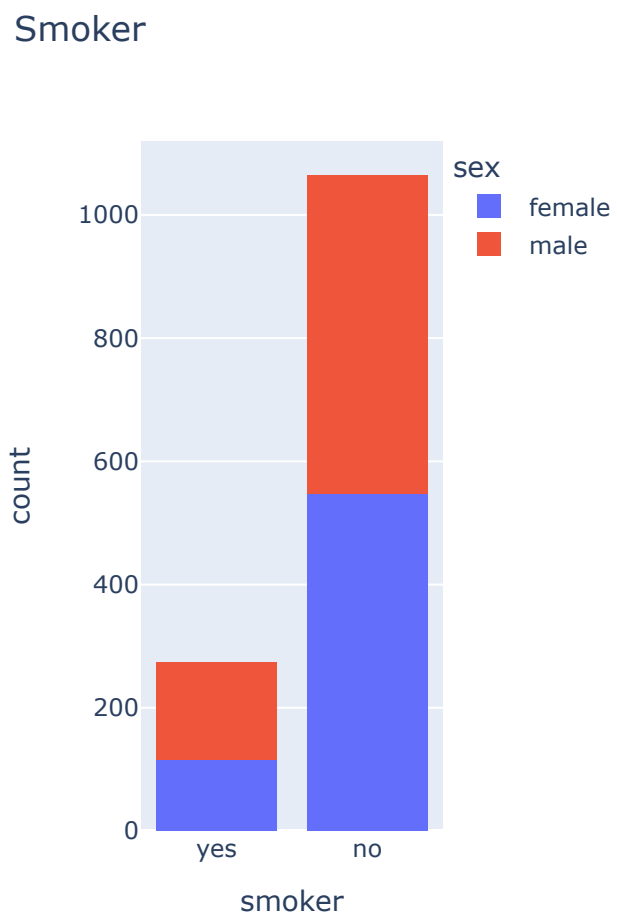
Overall, this correlation heatmap provides useful insights into the relationships between the features in the health insurance dataset. It can be used to guide further

> analysis and modeling of the data.

Distribution of smokers based on sex

```
import plotly.express as pxt
```

```
pxt.histogram(df_original, x='smoker', color='sex', title='Smoker')
```



INFERENCE

1. The above graph shows the distribution of the number of people who smoke categorised based on their sex.
2. There are more umber of people who don't smoke.

Distribution of charges based on bmi and sex

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

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

INFERENCE

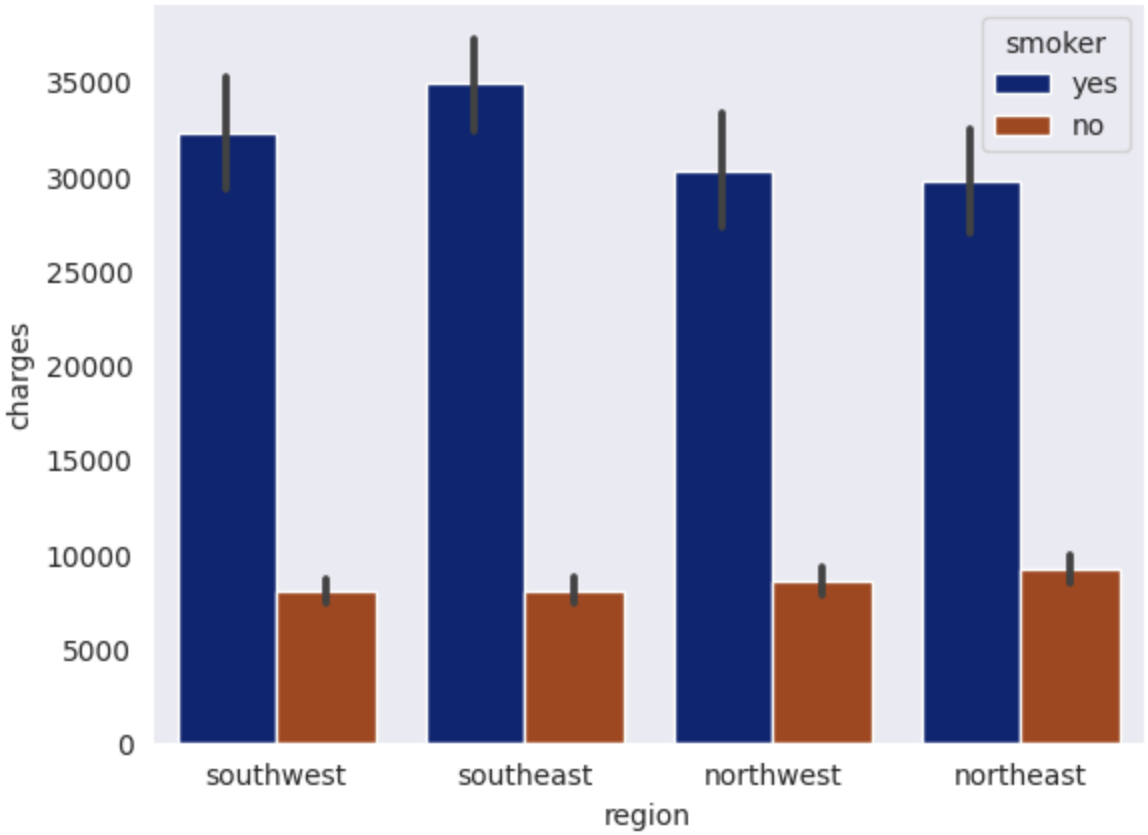
> As the bmi increases, the insurance charge also increases as shown above.

sex

Distribution of charges based on region and number of smokers in that region

```
sns.barplot(data = df_original,x = 'region',y = 'charges',hue = "smoker")
```

```
<Axes: xlabel='region', ylabel='charges'>
```

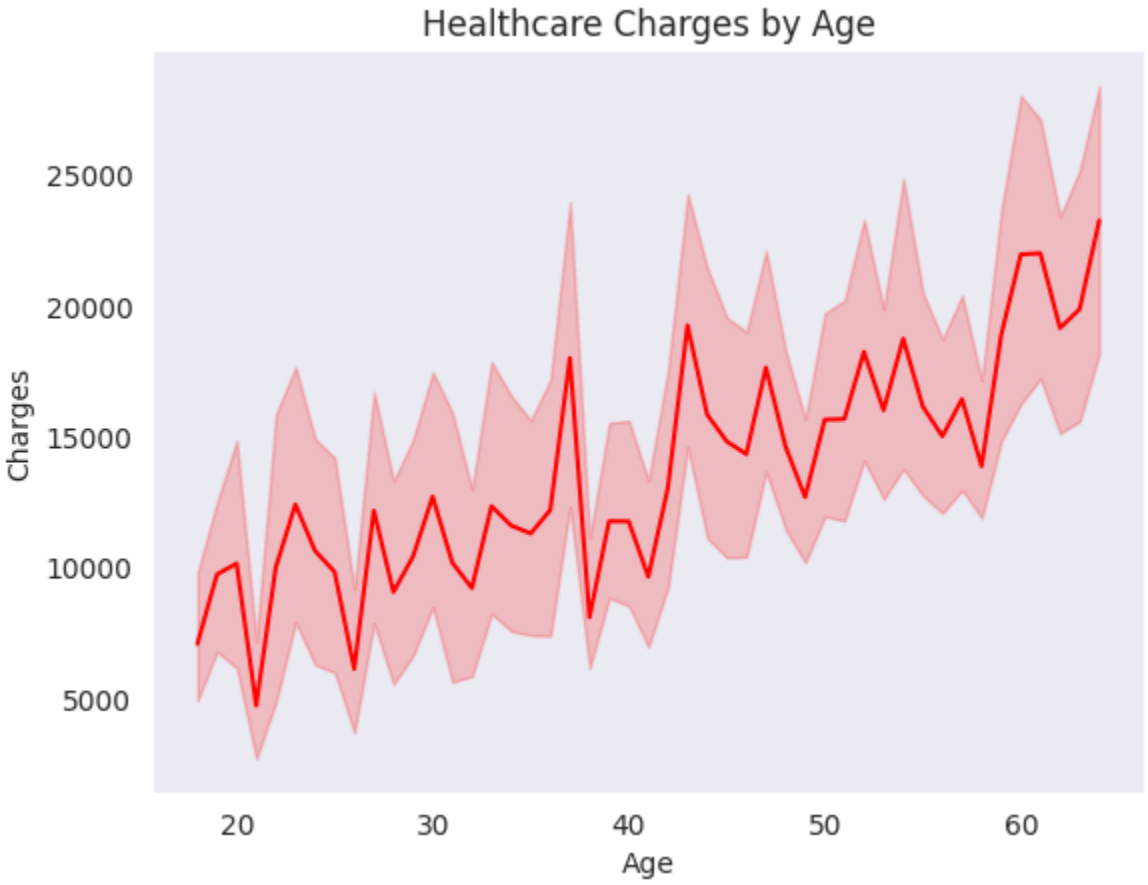


INFERENCE

> Southeast region has the most number of smokers, followed by Southwest, Northwest and Northeast

Distribution of charges based on age

```
# Set the plot style and background color
sns.set_style("dark")
sns.set_palette("dark")
#plt.rcParams['figure.facecolor'] = 'black'

# Create the plot
ax = sns.lineplot(data=df_original, x='age', y='charges', color='red')

# Set the plot title and axis labels
ax.set_title("Healthcare Charges by Age")
ax.set_xlabel("Age")
ax.set_ylabel("Charges")

# Remove the top and right spines
sns.despine()

# Show the plot
plt.show()
```

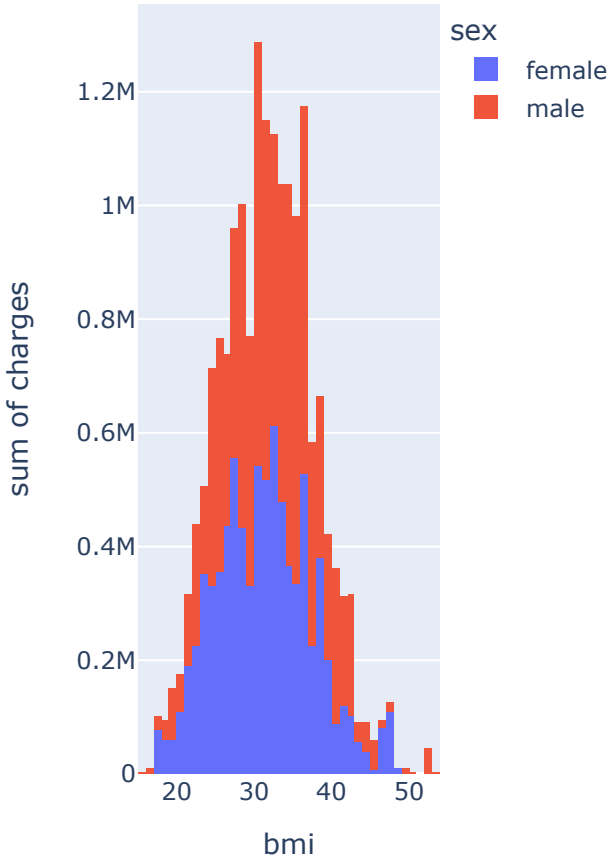


INFERENCE

> With increase in age the healthcare charges also increases as seen in the above ploy.

Distribution of charges with increase in BMI for males and females

```
pxt.histogram(df_original,x='bmi',y = 'charges',color = 'sex')
```



INFERENCE

> With increase in bmi the charges increases for both male and female, However the males are charged more compared to the females.

PAIR PLOTS

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

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1507: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
Ignoring `palette` because no `hue` variable has been assigned.
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:1609: UserWarning:
```

INFERENCE

> From this pairplot, we can make several observations:
>
> There is a clear separation between smokers and non-smokers in the plot, with smokers tending to have higher healthcare costs, higher BMIs, and lower ages.
>
> There is a positive linear relationship between age and charges, indicating that older individuals tend to have higher healthcare costs.
>
> There is a positive linear relationship between bmi and charges, indicating that individuals with higher BMIs tend to have higher healthcare costs.
>
> There is a weak positive relationship between children and charges, indicating that individuals with more children tend to have slightly higher healthcare costs.

DATA PREPROCESSING

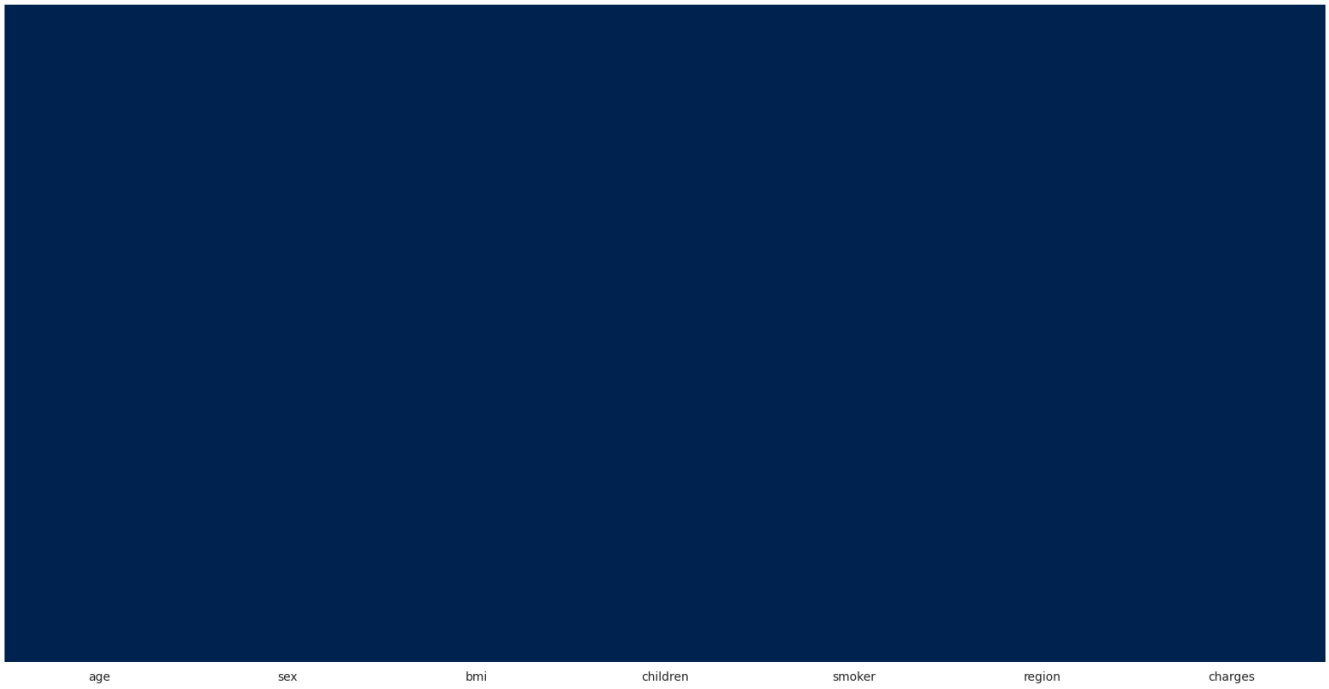
CHECK THE NULL VALUES

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

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

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

```
<Axes: >
```




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

```
column : age has 0 null values ---> 0.0 % of nulls
column : sex has 0 null values ---> 0.0 % of nulls
column : bmi has 0 null values ---> 0.0 % of nulls
column : children has 0 null values ---> 0.0 % of nulls
column : smoker has 0 null values ---> 0.0 % of nulls
column : region has 0 null values ---> 0.0 % of nulls
column : charges has 0 null values ---> 0.0 % of nulls
```

INFERENCE

> From the above analysis, we can see that none of the columns have nulls and therefore no imputation is required.

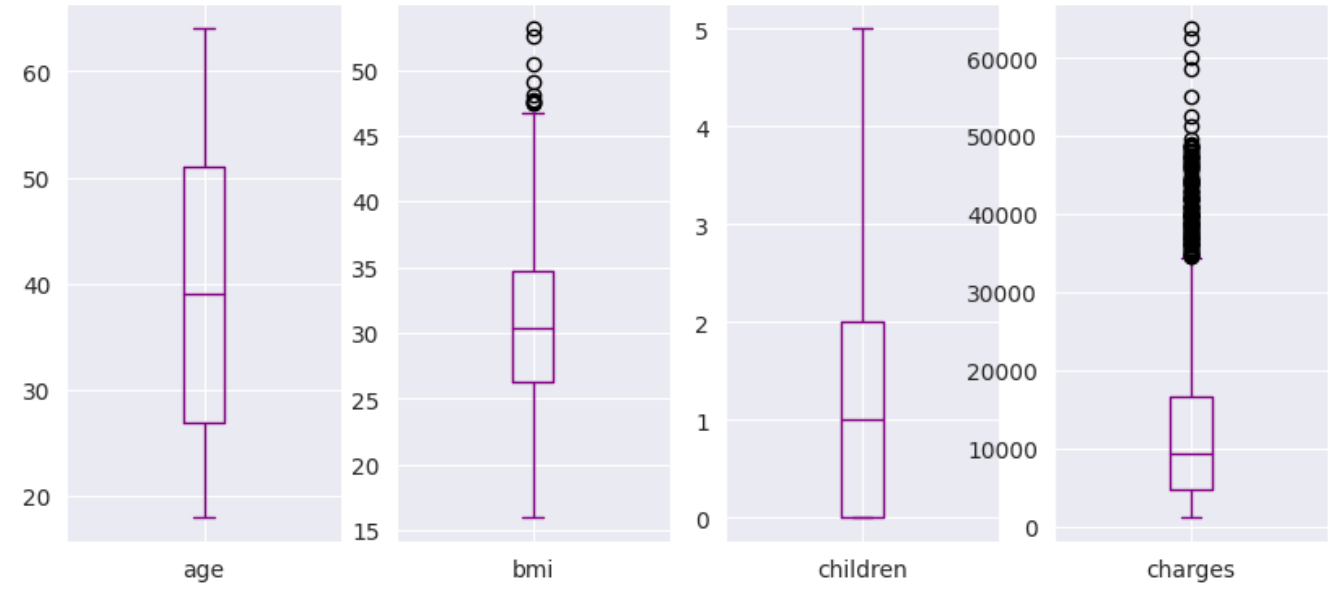
HANDLING OUTLIERS

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


plt.figure(figsize=(10,30))
numeric_columns = list(df_original.columns)
print(numeric_columns)
numeric_columns.remove("sex")
numeric_columns.remove("smoker")
numeric_columns.remove("region")

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

```
['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
```



INFERENCE

1. From the above graph we can see that charges has significant number of outliers, we further handle the outliers using the Tukey method

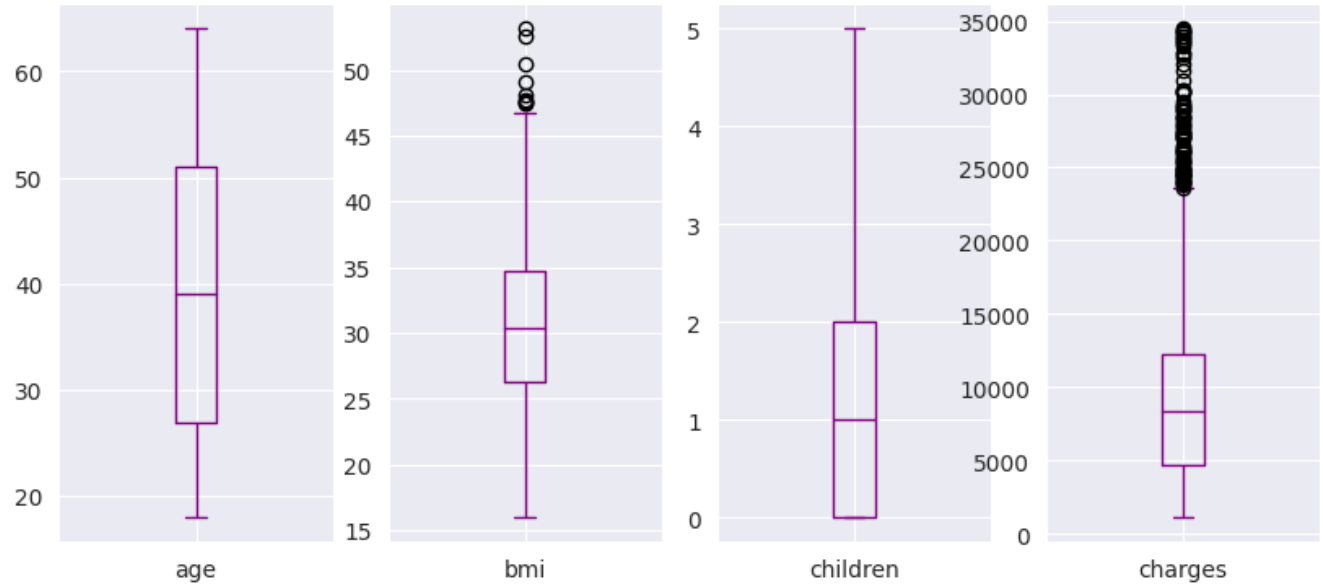
HANDLING OUTLIERS FOR THE COLUMN CHARGES

```
df_outliers_original = df_original

# Calculate the interquartile range (IQR)
Q1 = df_outliers_original.charges.quantile(0.25)
Q3 = df_outliers_original.charges.quantile(0.75)
IQR = Q3 - Q1

# Identify the outliers using the Tukey method
outliers = df_outliers_original[(df_outliers_original.charges < Q1 - 1.5 * IQR) | (df

# Replace the outliers with the median value of the non-outliers
non_outliers = df_outliers_original[(df_outliers_original.charges >= Q1 - 1.5 * IQR)
median = non_outliers.charges.median()
df_outliers_original.loc[outliers.index, 'charges'] = median
```

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

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


plt.figure(figsize=(10,30))
numeric_columns = list(df_outliers_original.columns)
print(numeric_columns)
numeric_columns.remove("sex")
numeric_columns.remove("smoker")
numeric_columns.remove("region")

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

```
['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
```



INFERENCE

We have then replaced the outliers with the median value of the non-outliers. We have identified the non-outliers using the same criterion as for the Tukey method, and calculated the median value of the non-outliers using the median() function from NumPy. We have used the loc method to update the values of the outliers in the charges column of the insurance dataframe.

By handling the outliers in this way, we can ensure that the charges variable is less susceptible to the influence of extreme values, and can improve the accuracy of any predictive models that are trained on the data. It is worth noting that there are other methods for handling outliers, and the choice of method will depend on the specific characteristics of the data and the requirements of the analysis.

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

```
df_outliers_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
```

```
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
df_encoder_original = df_outliers_original

# perform one hot encoding for smoker
df_encoder_original = pd.get_dummies(df_encoder_original, columns=['smoker'], prefix=
# perform one hot encoding for smoker
#df_encoder_original = pd.get_dummies(df_encoder_original, columns=['sex'], prefix='s

# display the updated dataframe
df_encoder_original.head()

# df_encoder_original_model = df_encoder_original
```

	age	sex	bmi	children	region	charges	smoker_yes
0	19	female	27.900	0	southwest	16884.92400	1
1	18	male	33.770	1	southeast	1725.55230	0
2	28	male	33.000	3	southeast	4449.46200	0
3	33	male	22.705	0	northwest	21984.47061	0
4	32	male	28.880	0	northwest	3866.85520	0

```
df_encoder_original.drop(['region'], axis=1, inplace= True)
df_encoder_original.drop(['sex'], axis=1, inplace= True)
df_encoder_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   bmi         1338 non-null   float64
 2   children    1338 non-null   int64
 3   charges     1338 non-null   float64
 4   smoker_yes  1338 non-null   uint8
dtypes: float64(2), int64(2), uint8(1)
memory usage: 43.2 KB
```

INFERENCE

Now we are done with data cleaning and data preprocessing. 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

Q-Q PLOT

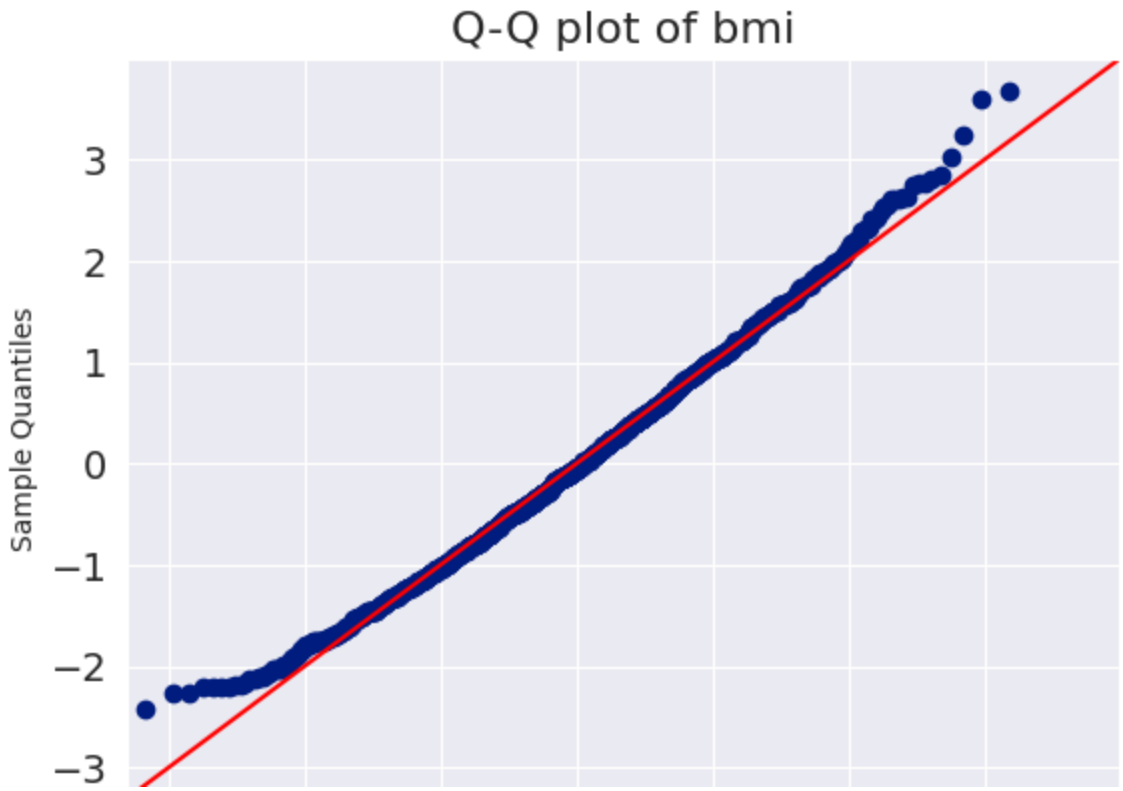
> In a Q-Q plot, the sample quantiles are plotted against the theoretical quantiles of the standard normal distribution on the x-axis and y-axis, respectively. If the sample follows a normal distribution, the Q-Q plot will show the points falling along a straight line. On the other hand, if the sample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

```
#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 --> age and 1
<Figure size 1000x2000 with 0 Axes>
```


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



INFERENCE

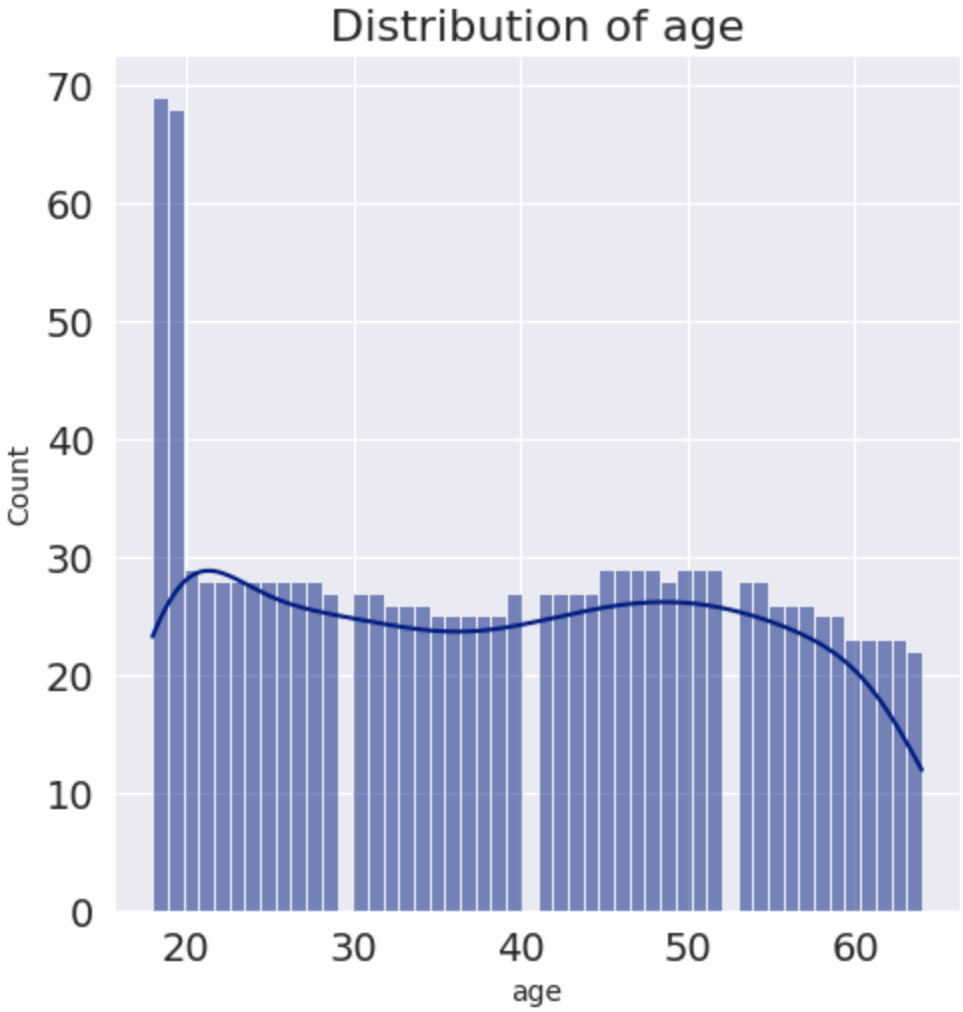
> From this Q-Q plot, we can see that the distribution of the charges variable deviates significantly from a normal distribution. There is a clear curvature in the plot, which suggests that the distribution of the charges variable is skewed. This is not surprising, given that healthcare costs are typically skewed towards high values.

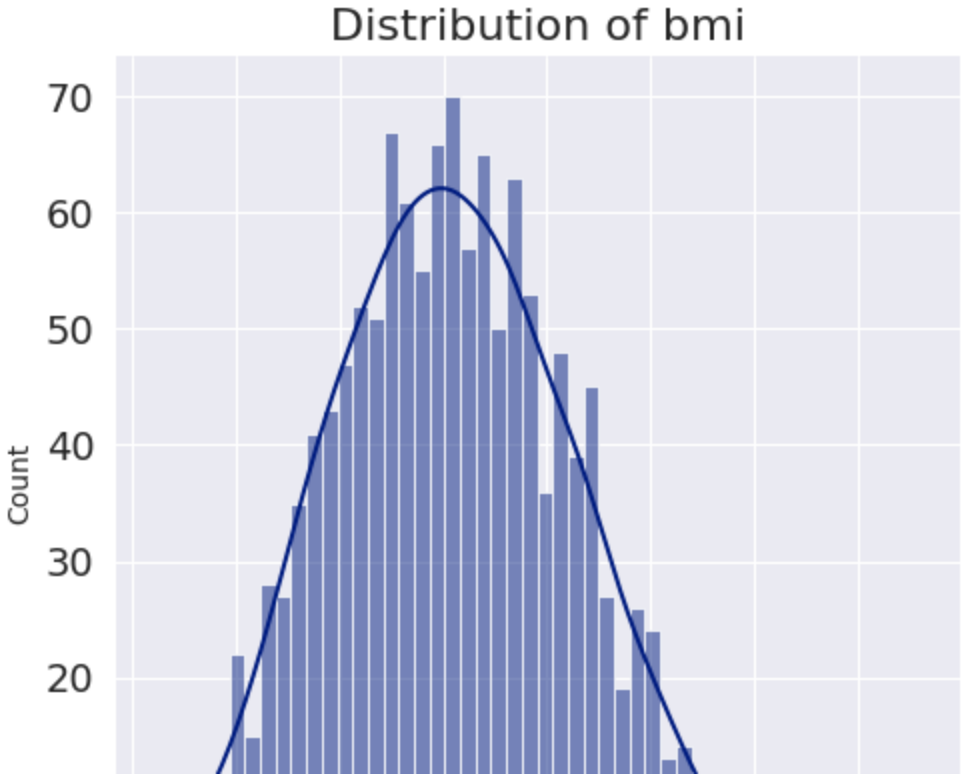
> Overall, this Q-Q plot provides a useful visualization of the distribution of the charges variable in the health insurance dataset. It can be used to guide the selection of appropriate statistical techniques and models for analyzing the data.

HISTOGRAM PLOT

```
#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)
```

```
<Figure size 2000x4000 with 0 Axes>
```

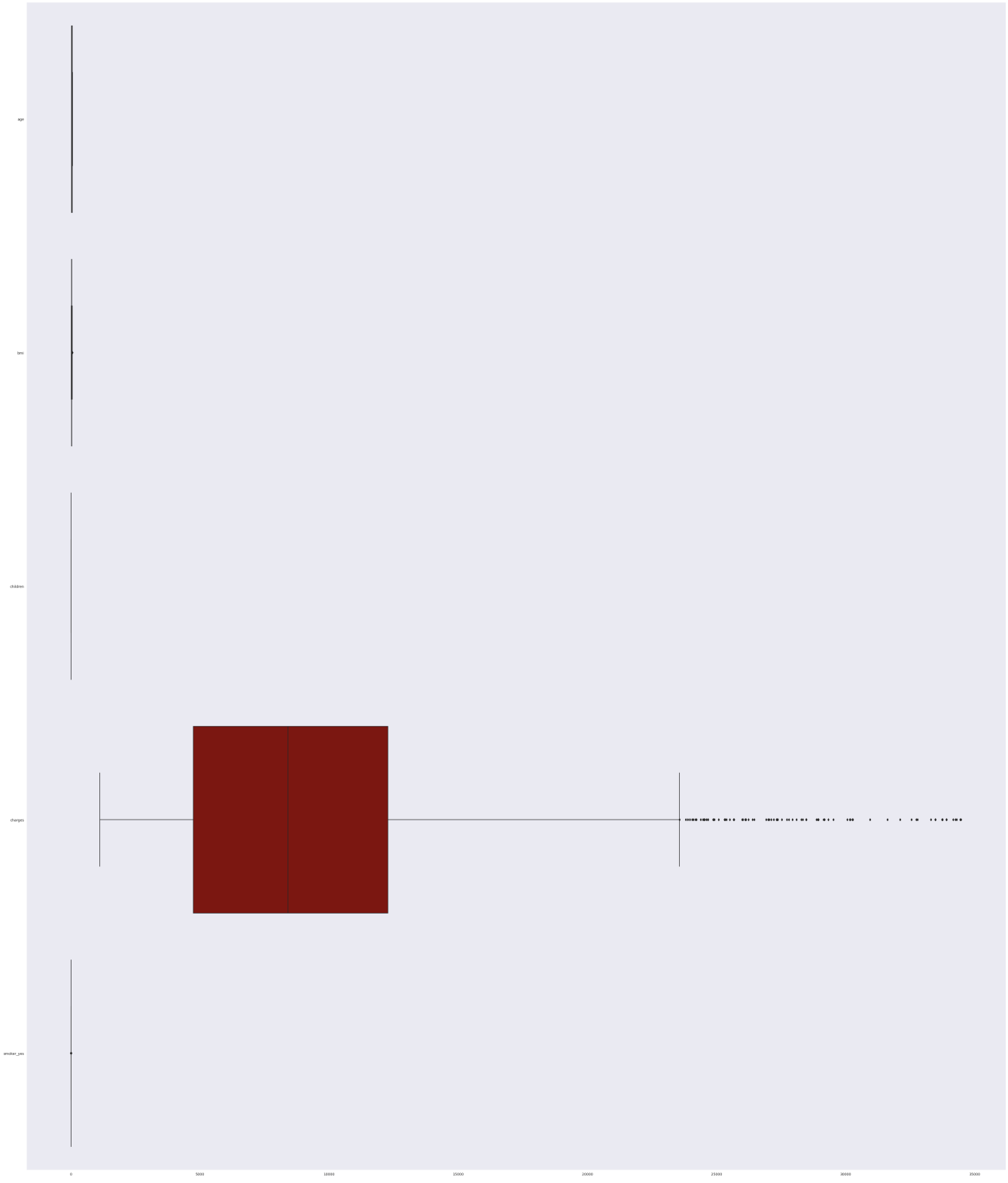




Checking the Ranges of the predictor variables and dependent variable


```
#taking only the clean table columns

columns = df_feature_original.columns
#checking the ranges of predictor and dependent variables

plt.figure(figsize=(50,60))
sns.boxplot(data=df_feature_original, orient = 'h')
```

```
<Axes: >
```


age
bmi
children
charges
smoker_yes
0
5000
10000
15000
20000
25000
30000
35000

INFERENCE

> the charges feature is highly skewed towards higher values, with many outliers towards the higher end.

> Overall, these box plots provide a useful summary of the distribution of each feature in the health insurance dataset. They can be used to identify outliers and to guide the selection of appropriate statistical techniques and models for analyzing the data.

PERFORMING NORMALIZATION USING MINMAX SCALAR

```
from sklearn.preprocessing import MinMaxScaler
df_normalized_original = df_feature_original.copy()

numeric_colums_values=list(df_normalized_original.columns)
numeric_colums_values

numeric_colums_values.remove('smoker_yes')
numeric_colums_values.remove('charges')

# perform normalization
scaler = MinMaxScaler()
df_normalized_original[numeric_colums_values] = scaler.fit_transform(df_normalized_or


df_normalized_original.head()
```

	age	bmi	children	charges	smoker_yes
0	0.021739	0.321227	0.0	16884.92400	1
1	0.000000	0.479150	0.2	1725.55230	0
2	0.217391	0.458434	0.6	4449.46200	0
3	0.326087	0.181464	0.0	21984.47061	0
4	0.304348	0.347592	0.0	3866.85520	0

```
df_normalized_final = df_normalized_original
df_normalized_final.head()
```

	age	bmi	children	charges	smoker_yes
0	0.021739	0.321227	0.0	16884.92400	1
1	0.000000	0.479150	0.2	1725.55230	0
2	0.217391	0.458434	0.6	4449.46200	0

```
df_normalized_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   age         1338 non-null   float64
 1   bmi         1338 non-null   float64
 2   children    1338 non-null   float64
 3   charges     1338 non-null   float64
 4   smoker_yes  1338 non-null   uint8  
dtypes: float64(4), uint8(1)
memory usage: 43.2 KB
```

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

	age	bmi	children	charges	smoker_yes
0	0.021739	0.321227	0.0	16884.92400	1
1	0.000000	0.479150	0.2	1725.55230	0
2	0.217391	0.458434	0.6	4449.46200	0
3	0.326087	0.181464	0.0	21984.47061	0
4	0.304348	0.347592	0.0	3866.85520	0

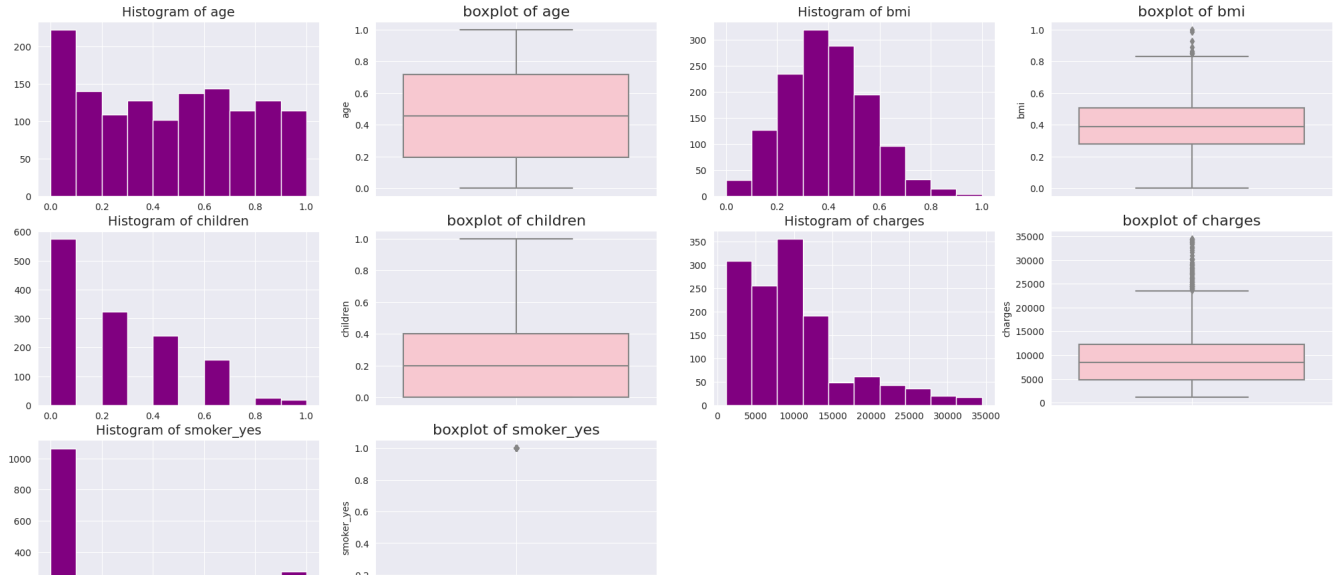
Correlation between the features after normalization

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

	age	bmi	children	charges	smoker_yes
age	1.000000	0.109272	0.042469	0.410797	-0.025019
bmi	0.109272	1.000000	0.012759	-0.080357	0.003750
children	0.042469	0.012759	1.000000	0.077884	0.007673
charges	0.410797	-0.080357	0.077884	1.000000	0.405802
smoker_yes	-0.025019	0.003750	0.007673	0.405802	1.000000

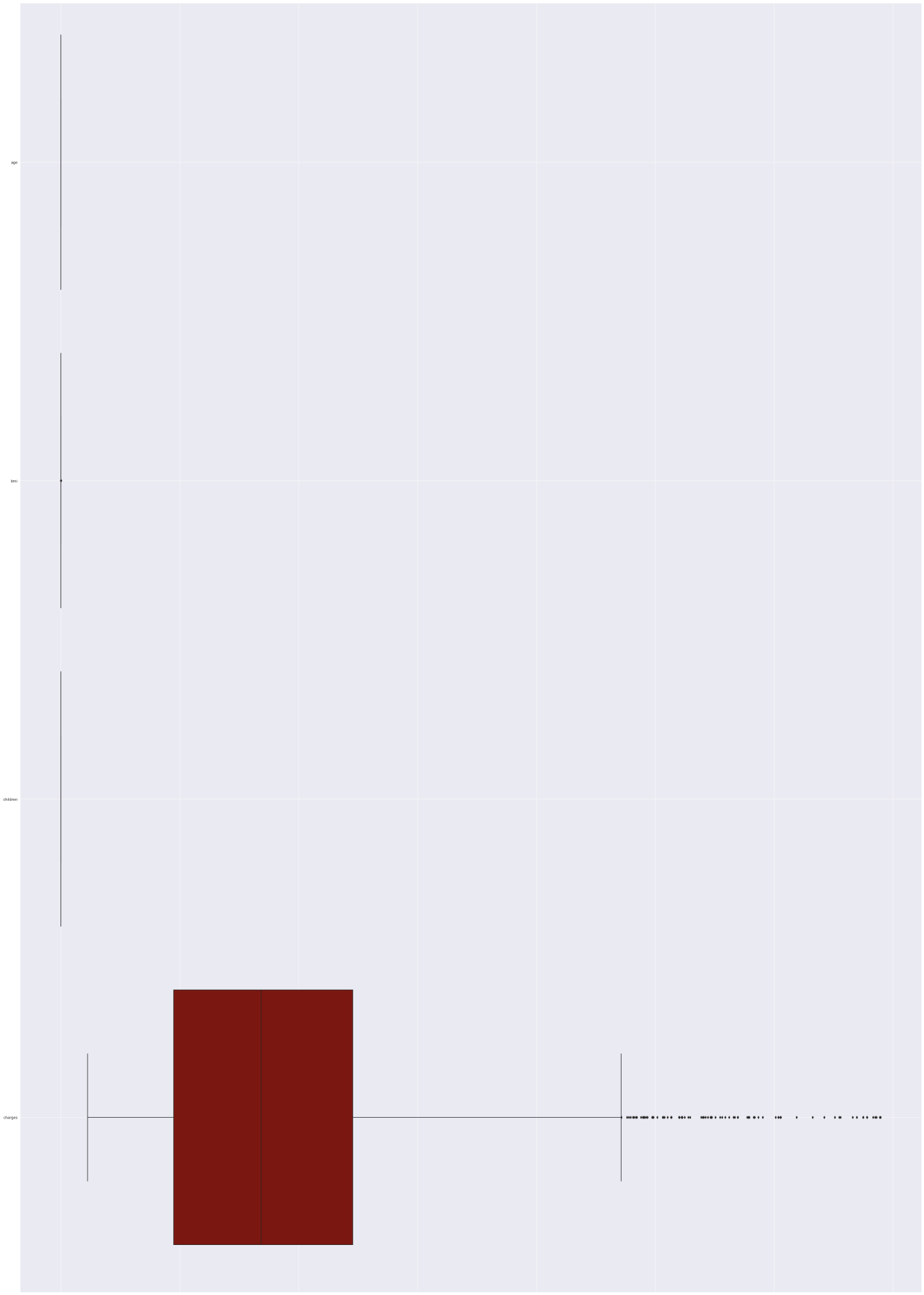
Visualizing Data Distribution after normalization

```
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()
```



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

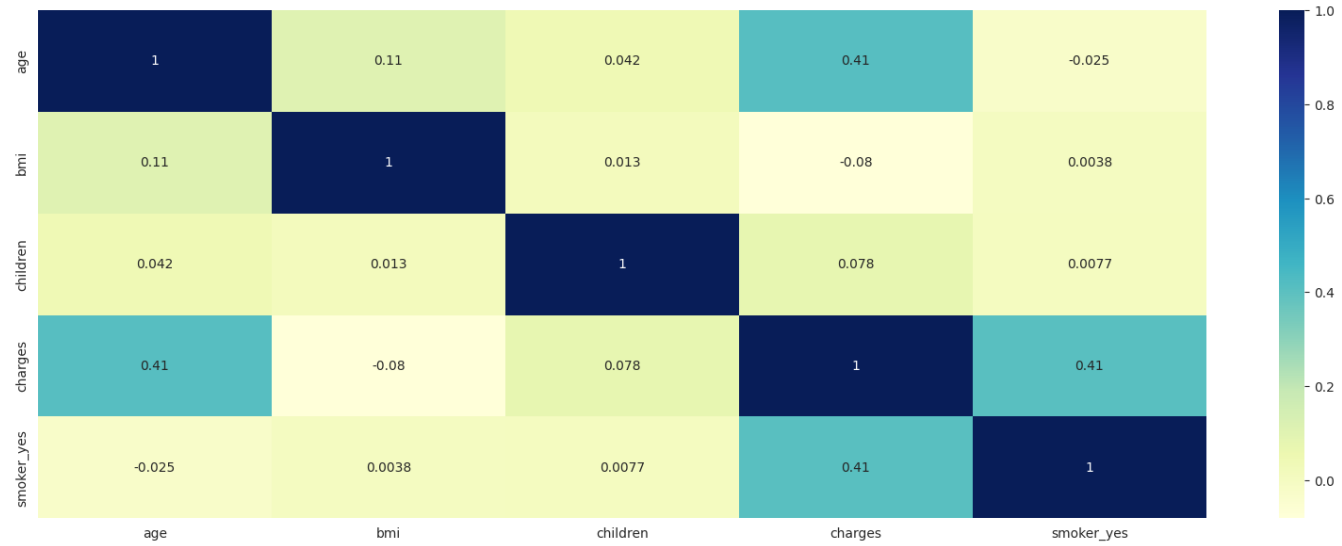
```
#checking the ranges of predictor and dependent variables

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


age
bmi
children
charges

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')
```

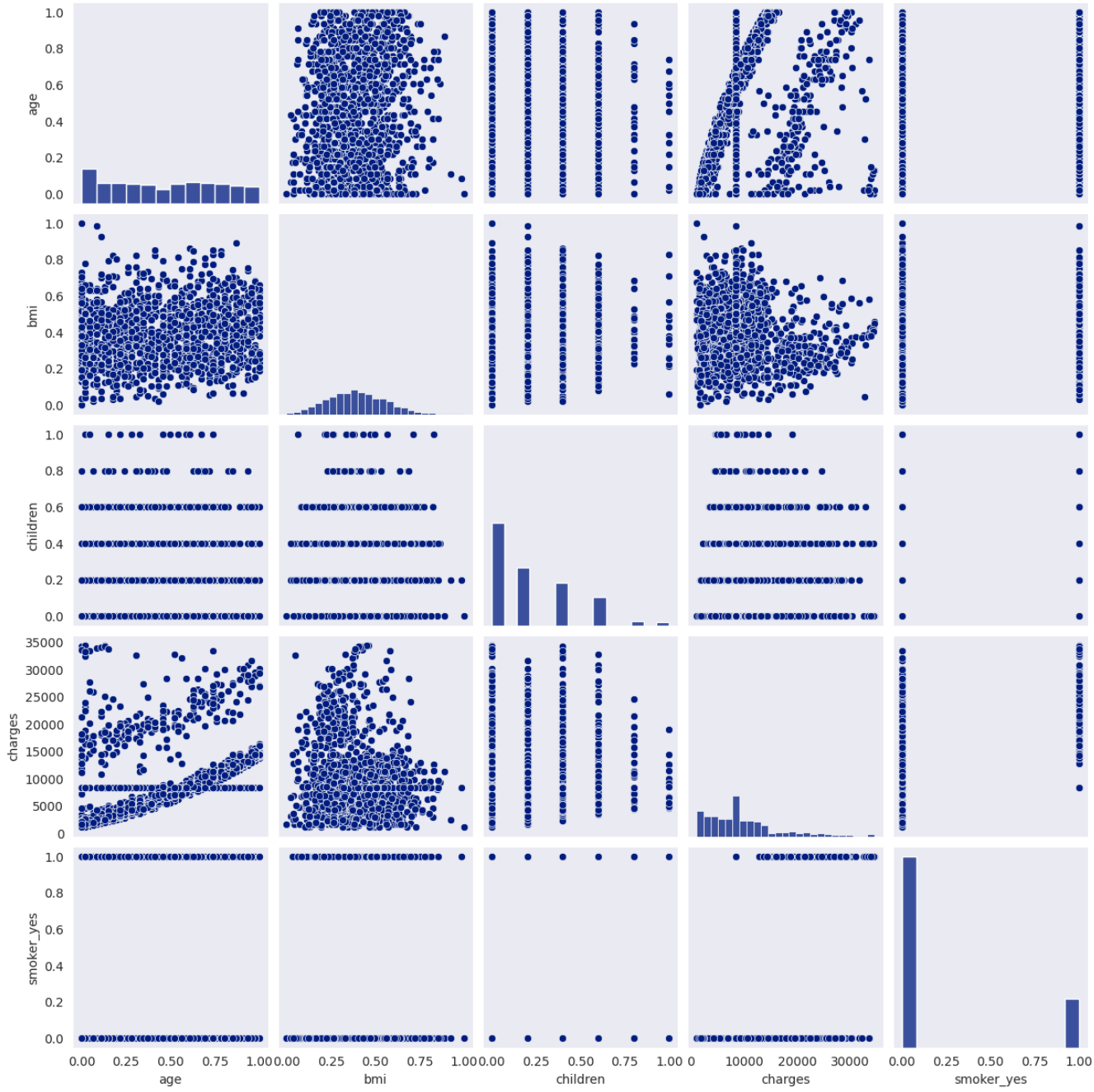
```
<Axes: >
```



Visualizing Pair plot of the correlation between features after normalization

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

```
<seaborn.axisgrid.PairGrid at 0x7f067eebc640>
```



INFERENCE

1. From this pair plot, we can see that there is a strong positive correlation between charges and age, as well as a moderate positive correlation between charges and bmi.
2. From this box plot, we can see that the normalized age, bmi, and charges variables have outliers towards higher values, while the children variable does not have any outliers.

```
df_normalized_final
```

	age	bmi	children	charges	smoker_yes
0	0.021739	0.321227	0.0	16884.92400	1
1	0.000000	0.479150	0.2	1725.55230	0
2	0.217391	0.458434	0.6	4449.46200	0
3	0.326087	0.181464	0.0	21984.47061	0
4	0.304348	0.347592	0.0	3866.85520	0
...	...	...	...	...	...
1333	0.695652	0.403820	0.6	10600.54830	0
1334	0.000000	0.429379	0.0	2205.98080	0
1335	0.000000	0.562012	0.0	1629.83350	0
1336	0.065217	0.264730	0.0	2007.94500	0
1337	0.934783	0.352704	0.0	29141.36030	1

1338 rows × 5 columns

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

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

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

model.summary()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   age         1338 non-null   float64
 1   bmi         1338 non-null   float64
 2   children    1338 non-null   float64
 3   smoker_yes  1338 non-null   uint8  
dtypes: float64(3), uint8(1)
memory usage: 32.8 KB
['age', 'bmi', 'children', 'smoker_yes']
```

OLS Regression Results

Dep. Variable:	charges	**R-squared (uncentered):**	0.765
Model:	OLS	**Adj. R-squared (uncentered):**	0.765
Method:	Least Squares	**F-statistic:**	1087.
Date:	Mon, 24 Apr 2023	**Prob (F-statistic):**	0.00
Time:	04:21:07	**Log-Likelihood:**	-13490.
No. Observations:	1338	**AIC:**	2.699e+04
Df Residuals:	1334	**BIC:**	2.701e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
age	1.218e+04	478.269	25.470	0.000	1.12e+04	1.31e+04
bmi	2856.3815	657.838	4.342	0.000	1565.873	4146.890
children	3704.4056	633.657	5.846	0.000	2461.333	4947.478
smoker_yes	7871.1601	386.879	20.345	0.000	7112.203	8630.117

Omnibus:	416.218	**Durbin-Watson:**	1.957
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1601.299
Skew:	1.464	**Prob(JB):**	0.00
Kurtosis:	7.489	**Cond. No.**	3.61

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

INFERENCE

> The model summary reveals that the model has an R-squared value of 0.812, indicating that the independent variables explain about 81.2% of the variance in the

> dependent variable.

> The p-values of the independent variables indicate that **age**, **BMI**, **children**, **smoker_yes**, as their p-values are less than 0.05.

BUILDING THE MODEL USING LINEAR REGRESSION

TRAIN, TEST SPLIT

```
from sklearn.model_selection import train_test_split


df_model_original = df_normalized_final.copy()

#Removing the insignificant features before modeling

df_model_original.info()
df = df_model_original.drop(['charges'], axis=1)
columns = list(df.columns)
X = df_model_original[columns]
y = df_model_original['charges']


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

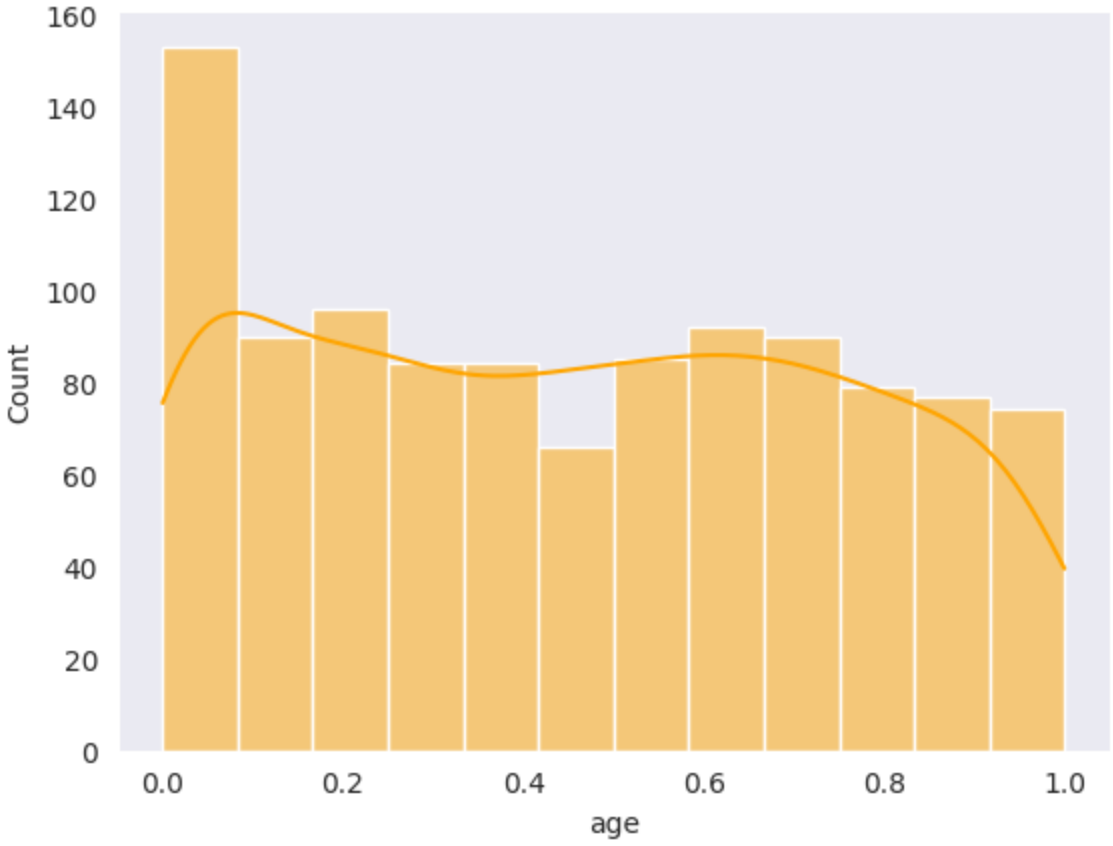
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   age         1338 non-null   float64
 1   bmi         1338 non-null   float64
 2   children    1338 non-null   float64
 3   charges     1338 non-null   float64
 4   smoker_yes  1338 non-null   uint8  
dtypes: float64(4), uint8(1)
memory usage: 43.2 KB
```

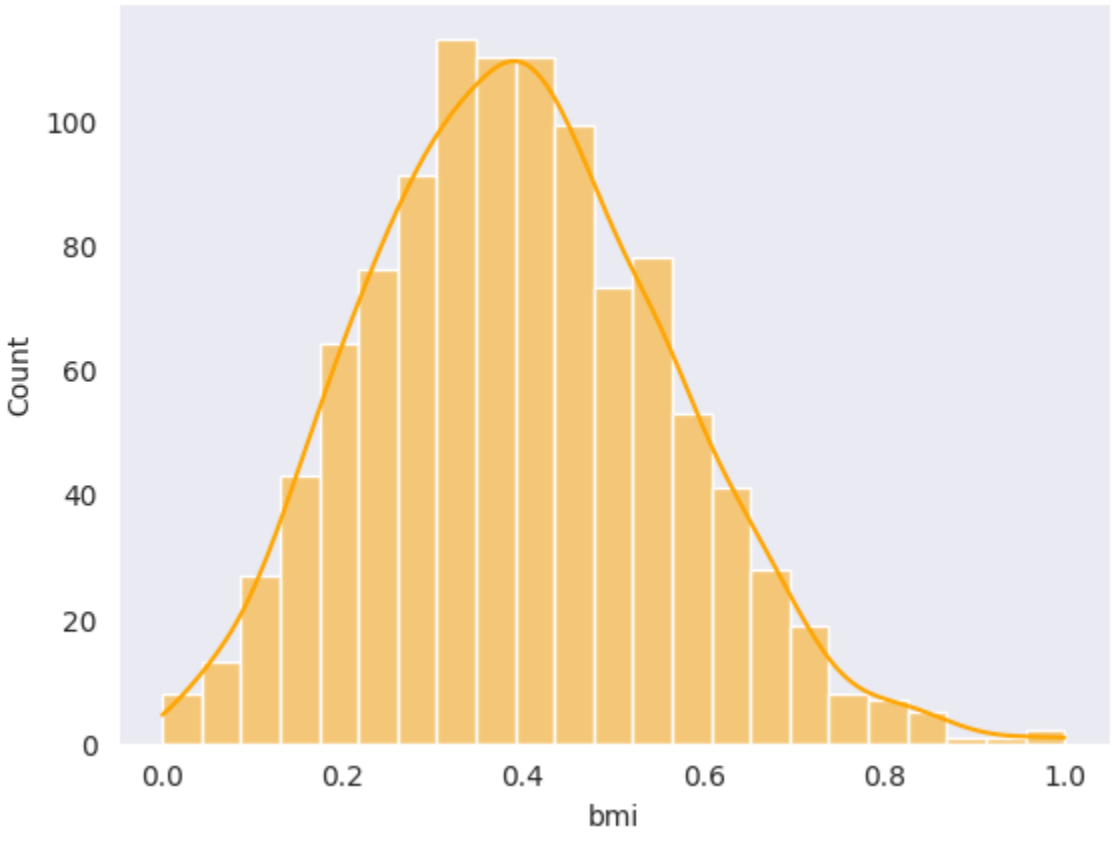
```
X_train.head()
```

	age	bmi	children	smoker_yes
216	0.760870	0.286252	0.0	0
731	0.760870	0.146355	0.2	0
866	0.000000	0.573850	0.0	0

DISTRIBUTION OF TRAINING DATA

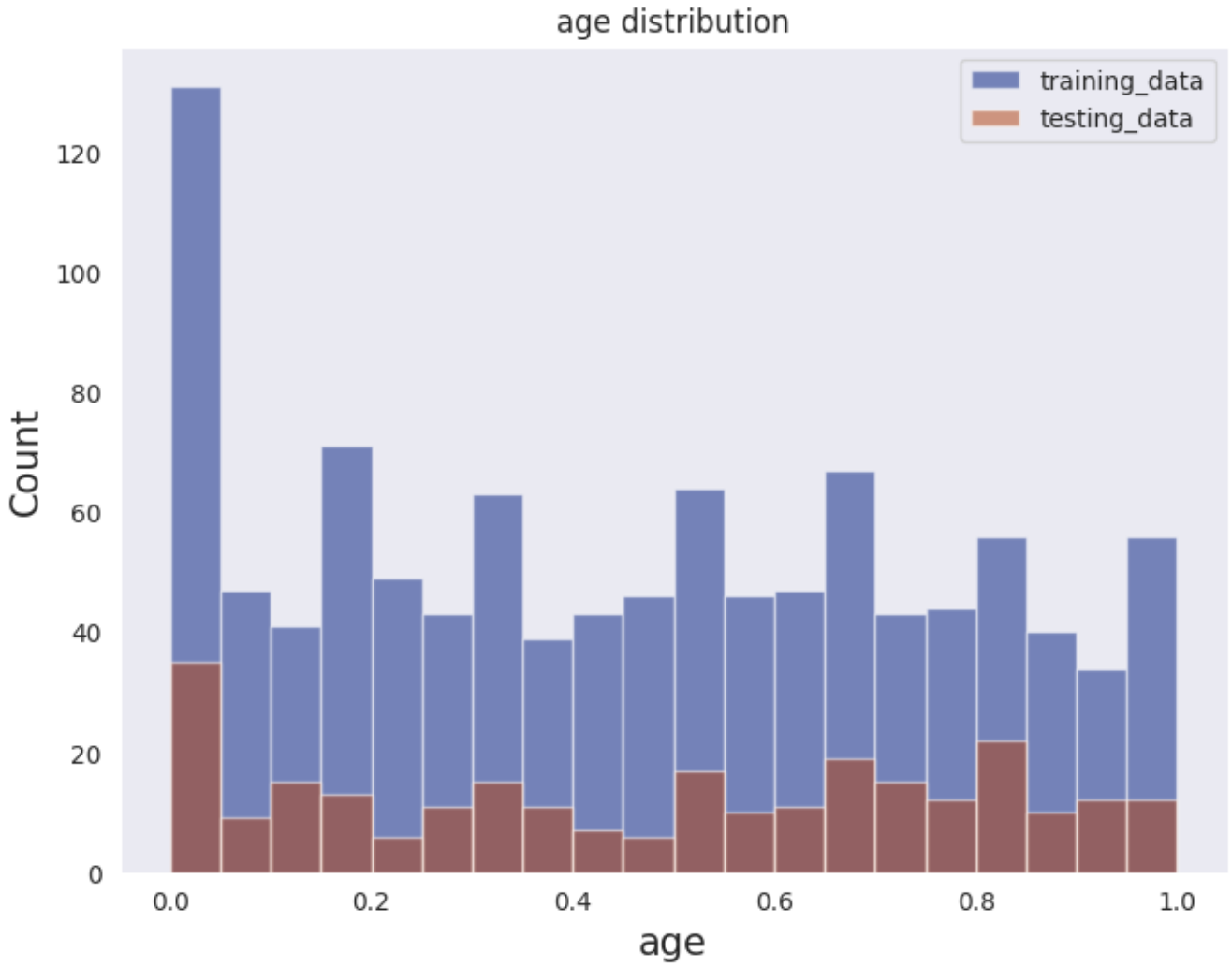
```
for i, col in enumerate(X_train.columns):
    plt.figure(i)
    sns.histplot(X_train[col], kde=True, color="orange")
```

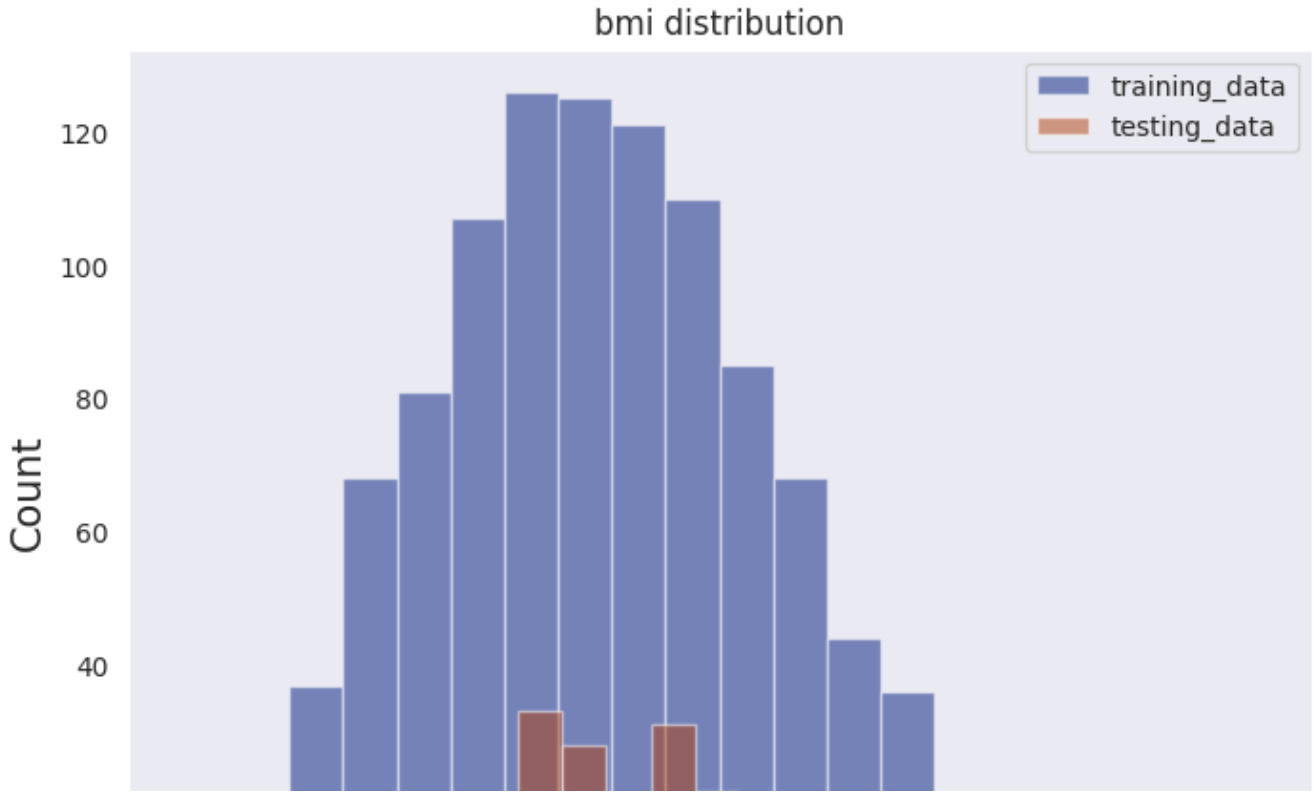

160
140
120
100
80
60
40
20
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
age


100
80
60
40
20
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
bmi

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()
```


age distribution
training_data
testing_data
Count
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0
age


bmi distribution
training_data
testing_data
Count
40
60
80
100
120

20

INFERENCE

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

Fit the model using the training data

400

```
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

LinearRegression()

Model Coefficients

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

array([9518.7241331 , -5461.72410498, 1540.81472224, 7359.57850385])

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

	Coeff
age	9518.724133
bmi	-5461.724105
children	1540.814722
smoker_yes	7359.578504

400

PREDICTIONS AND EVALUATION

200

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)#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]:  3569.87
Mean Square Error [MSE]:  30937078.25
Root mean Square Error [RMSE]:  5562.111

Coefficient of Determination : 0
R^2 score on training set = 0.35815127825639625
Mean Absolute Error [MAE]:  3569.87
Mean Square Error [MSE]:  30937078.25
Root mean Square Error [RMSE]:  5562.111

Coefficient of Determination : 0
R^2 score on training set = 0.35815127825639625
```

TEST DATASET

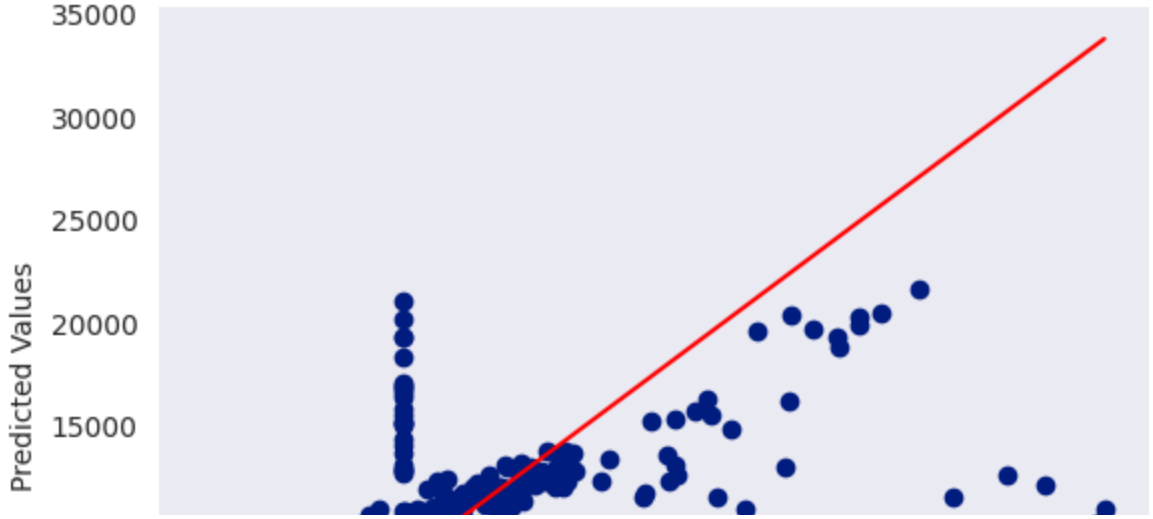
```
#PREDICTION ON THE TRAINING SET
#EVALUATE THE MODEL

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]:  3316.22
Mean Square Error [MSE]:  26824481.33
Root mean Square Error [RMSE]:  5179.236

Coefficient of Determination : 0
R^2 score on training set = 0.37645413821101037
```

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 0x7f067c59ff10>]
```



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

RESIDUALS

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

> Residual = Observed value – Predicted value

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

> 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.

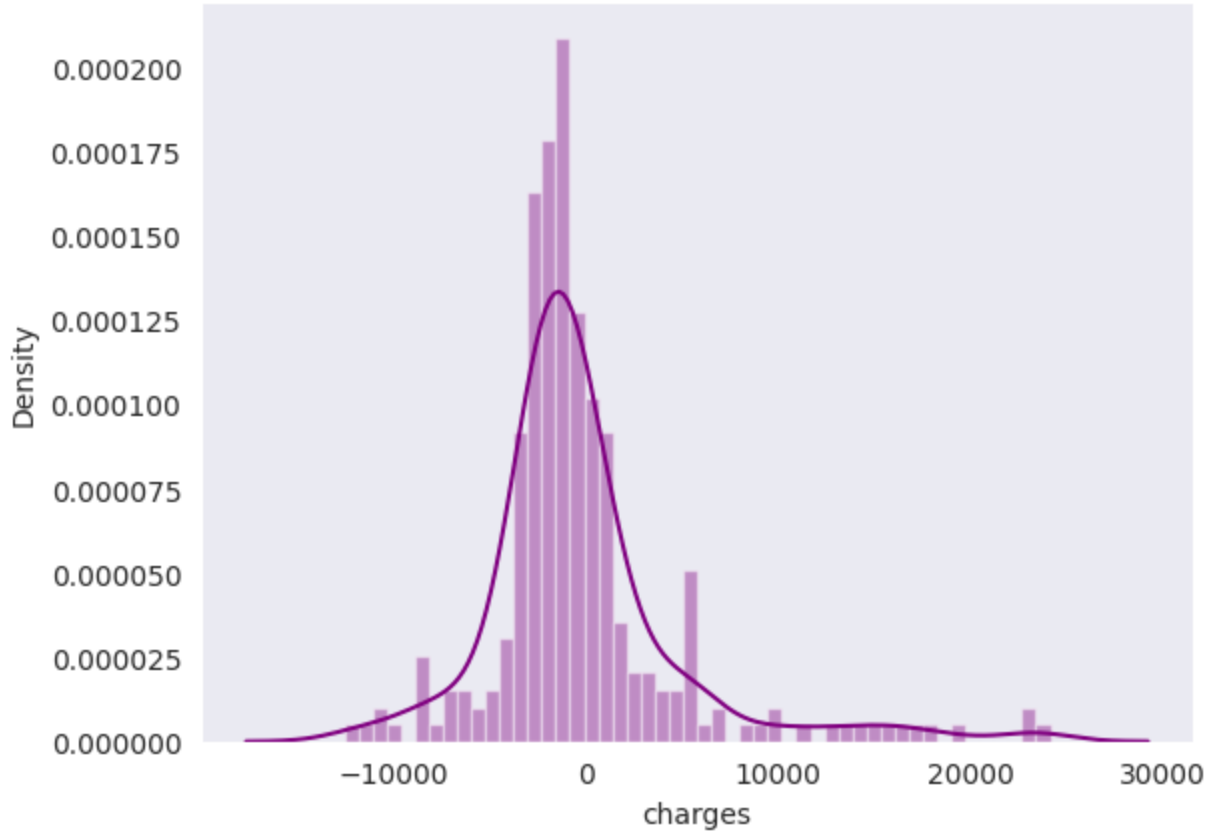
REFERENCES

> 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')
```

```
<ipython-input-554-b8771af6b358>:3: UserWarning:


`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='charges', 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
0.4278 ± 0.1013	age
0.3058 ± 0.1007	smoker_yes
0.0362 ± 0.0171	bmi
0.0104 ± 0.0076	children

CALCULATING MSE AND VARIANCE USING USER-DEFINED FUNCTION AND COMPARING THE VALUES WITH 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/

FUNCTION DEFINED TO IMPLEMENT MSE AND VARIANCE MATHEMATICALLY AND COMPARE IT WITH THE VALUE OBTAINED USING SCIKIT BUILT-IN FUNCTION

```
def statistical_measures(y_original, y_predictions):

  length = len(y_original)

  residuals = y_original-y_predictions
  residual_mean = residuals.mean()


  print("-------------------------------------------------------------------------------
  print("\nMEAN SQUEARE ERROR (MSE)")
  print("-------------------------------------------------------------------------------
  print("MATHEMATICAL FORMULA FOR FINDING MSE:")
  print('''
╔═══════════════════════════════════════════════╗
║  MSE = 1/n * ∑(i=1 to n) (yi - ŷi)^2          ║
╚═══════════════════════════════════════════════╝
''')


  #FINDING MSE USING THE MATHEMATICAL FORMULA
  mse_formula = round((pow((residuals),2)).sum()/length,2)
  print("MSE VALUE USING MATHEMATICAL FORMULA --> ", mse_formula )

  #FINDING MSE USING THE SCIKIT INBUILT FORMULA
  print('MSE VALUE USING SCIKIT FUNCTION: ', round(metrics.mean_squared_error(y_origi

  print("-------------------------------------------------------------------------------
  print("-------------------------------------------------------------------------------

  print("\n RESIDUAL VARIANCE")
  print("-------------------------------------------------------------------------------
  print("MATHEMATICAL FORMULA FOR FINDING RESIDUAL VARIANCE:")
  print('''
╔═══════════════════════════════════════════════════╗
║  Residual Variance = 1/(n-k-1) * ∑(i=1 to n) (yi - ŷi)^2  ║
╚═══════════════════════════════════════════════════╝
''')


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

  #FINDING VARIANCE USING THE MATHEMATICAL FORMULA
  var_formula = round((pow((residuals - residual_mean),2)).sum()/length,5)
```

```
  print("\nVARIANCE USING MATHEMATICAL FORMULA --> ", var_formula )

  #FINDING VARIANCE USING THE SCIKIT INBUILT FORMULA
  var = round(statistics.variance(list(residuals)),5)
  print("VARIANCE USING SCIKIT FUNCTION --> ", var)
  print("--------------------------------------------------------------------------------")
```

FUNCTION DEFINED TO PLOT THE RESIDUAL

```
def plot_residual(y_original,y_predictions):


  residual =  y_original - y_predictions

  # Importing the required libraries
  import matplotlib.pyplot as plt
  import seaborn as sns

  # SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
  print("SCATTER PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES")
  fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4))
  sns.scatterplot(x=y_original, y=y_predictions, ax=ax1)
  ax1.plot([y_original.min(), y_original.max()], [y_original.min(), y_original.max()]
  ax1.set_xlabel('True Values')
  ax1.set_ylabel('Predicted Values')

  # RESIDUAL PLOT OF THE REAL VALUES VERSUS THE PREDICTED VALUES
  sns.distplot((residual), bins=50, color='orange', ax=ax2)
  ax2.set_xlabel('Residual')
  ax2.set_ylabel('Frequency')

  # Display the plots
  plt.tight_layout()
  plt.show()
```

Implementation on TRAINING DATA

```
y_predictions = lm.predict(X_train)
statistical_measures(y_train, y_predictions)
```

```
    --------------------------------------------------------------------------------

    MEAN SQUEARE ERROR (MSE)
```