# Unveiling the Secrets of Global Life Expectancy

## 1. Project Description

### Introduction

Life expectancy is a critical indicator that reflects the overall health and well-being of a population. It measures the average number of years a person is expected to live, based on current mortality rates. This metric is influenced by factors such as healthcare access, education, economic resources, and social conditions. The goal of this project is to identify the key determinants that influence life expectancy, particularly in low-income countries, where resource allocation plays a crucial role in improving health outcomes.

### Objectives

The primary objective of this project is to identify and rank the most significant factors affecting life expectancy. By applying data-driven insights, we aim to uncover these critical determinants and develop predictive models to forecast life expectancy. The insights gained will inform policy-making and help optimize resource allocation across healthcare, education, and economic development, with a particular focus on regions facing economic challenges.

### Methodology

#### Step 1: Load the Dataset

The dataset used for this project is the Life Expectancy Data from the World Health Organization (WHO), which covers 193 countries over the period from 2000 to 2015. This dataset, available for download from Kaggle, includes variables such as healthcare expenditure, education, GDP, and disease prevalence. We imported this data into a Python environment using Pandas for further processing.

#### Step 2: Clean the Dataset

Data cleaning is a crucial step to ensure the dataset is free from errors, inconsistencies, and missing values. We applied several strategies for handling missing data:

- **Imputation with Closest Three-Year Average:** Missing values for specific years were filled using the average of values from the previous and next years.
- **Imputation with Regional Averages:** When data was missing for all years of a country, we used the regional averages (e.g., Asia, Africa) to fill in the gaps.

- **Dropping Rows with Excessive Missing Values:** Countries with more than four missing values across all columns were removed.

Additionally, we cleaned the column names, removing extra spaces, and dropped irrelevant columns such as 'thinness 5-9 years', 'Total expenditure', 'infant deaths', and 'Country'. We also mapped the 'Status' column (indicating whether a country is developed or developing) to numerical values (0 for Developed, 1 for Developing) for model compatibility. This left us with a cleaned dataset ready for analysis.

**Step 3: Explore the Dataset**

In this step, we explored the dataset to understand the relationships between different variables and life expectancy. We created scatter plots to visualize how life expectancy correlates with all the remaining variables, such as adult mortality, income composition, and education.

We, for example, found a negative correlation between adult mortality and life expectancy, meaning that higher adult mortality rates are associated with shorter life spans. Visualizations also highlighted a significant gap in life expectancy between developed and developing countries. Histograms and density plots helped us assess the distribution of life expectancy values across different regions and years.

Furthermore, we computed a correlation matrix to identify which features were strongly correlated with life expectancy. This allowed us to pinpoint variables that had the most significant impact.

**Step 4: Feature Selection**

Feature selection helps to identify the most important predictors for the target variable, life expectancy, and reduces the dimensionality of the dataset. We used mutual information to quantify the relationship between each feature and the target variable. Features with higher mutual information are considered more important for predicting life expectancy.

We identified the following key features as the most important for our predictive model:

- Adult Mortality
- Income Composition of Resources
- Thinness 1-19 Years
- Schooling
- BMI
- HIV/AIDS
- Under-Five Deaths
- GDP

On the other hand, features like Alcohol Consumption, Polio Immunization, Diphtheria Vaccination, and Measleswere found to be less impactful in predicting life expectancy.

**Step 5: Model Creation**

After feature selection, we moved forward with building a predictive model for life expectancy. We compared several regression models, including Gradient Boosting, Linear Regression, XGBoost, and Support Vector Regression (SVR). We evaluated each model based on key performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Cross-Validation RMSE (CV-RMSE), and R-squared (R2).

Through this evaluation, we found that the Random Forest Regressor consistently outperformed the other models across all metrics. Due to its superior performance, we decided to base our final predictive model on Random Forest. The next steps included training and fine-tuning the model to optimize its performance for accurate life expectancy predictions.

**Conclusion**

This project successfully identified and ranked the most significant factors affecting life expectancy. By comparing various regression models, we determined that the Random Forest Regressor provided the best predictive performance. The insights from this study can inform policy decisions and improve resource allocation in low-income regions, driving targeted interventions to enhance life expectancy. By focusing on key factors and leveraging predictive modeling, we can improve life expectancy and quality of life globally, particularly in resource-constrained countries.

## 2. Code

### Step 1: Data Acquisition and Initial Loading

I. Load the dataset.

```
import kagglehub
import os

# Download latest version
path = kagglehub.dataset_download("kumarajarshi/life-expectancy-who")

print("Path to dataset files:", path)

import pandas as pd

# Construct the file path using os.path.join
file_path = os.path.join(path, 'Life Expectancy Data.csv')

# Load the dataset into a Pandas DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
df.head()
```

| | Country | Year | Status | Life expectancy | Adult Mortality | infant deaths | Alcohol | percentage expenditure | Hepatitis B | Measles | ... | Polio | Total expenditure | Diphtheria | HIV/AIDS | GDP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Afghanistan | 2015 | Developing | 65.0 | 263.0 | 62 | 0.01 | 71.279624 | 65.0 | 1154 | ... | 6.0 | 8.16 | 65.0 | 0.1 | 584.259210 |
| 1 | Afghanistan | 2014 | Developing | 59.9 | 271.0 | 64 | 0.01 | 73.523582 | 62.0 | 492 | ... | 58.0 | 8.18 | 62.0 | 0.1 | 612.696514 |
| 2 | Afghanistan | 2013 | Developing | 59.9 | 268.0 | 66 | 0.01 | 73.219243 | 64.0 | 430 | ... | 62.0 | 8.13 | 64.0 | 0.1 | 631.744976 |
| 3 | Afghanistan | 2012 | Developing | 59.5 | 272.0 | 69 | 0.01 | 78.184215 | 67.0 | 2787 | ... | 67.0 | 8.52 | 67.0 | 0.1 | 669.959000 |
| 4 | Afghanistan | 2011 | Developing | 59.2 | 275.0 | 71 | 0.01 | 7.097109 | 68.0 | 3013 | ... | 68.0 | 7.87 | 68.0 | 0.1 | 63.537231 |

5 rows x 22 columns

II. Check the columns information.

```
df.info()
```

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

## Step 2: Data Cleaning and Missing Values Imputation

I. Check if there are any null values in any of the columns.

```
print(df.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
```

II. Correct the columns name format.

```
# Clean column names (strip leading/trailing whitespaces, if any)
df.columns = df.columns.str.strip()
# apply the missing value handling and cleaning strategies
df_cleaned = df.copy()  # Create a copy of the original DataFrame to keep it intact
```

III. Fill the missing values.

*The data had some missing values. A few strategies for filling in missing values were applied.*

1. *Filling data with the closest three-year average. If a specific country had a missing value in any year, the data was filled with the closest three-year average.*
2. *Filling data with the average of the Region. If a specific country was missing values for all years, the data was filled with the average of the Region (e.g. Asia, Africa, European Union, etc.)*

*Data is adjusted and the missing values are filled. Countries that were missing more than 4 data columns were omitted from the database. Examples of these countries are Sudan, South Sudan, and North Korea.*

```python
import pandas as pd
import numpy as np


# List of columns with missing data
columns_with_missing_data = [
    'Life expectancy', 'Adult Mortality', 'Alcohol', 'GDP', 'Population', 'Hepatitis B',
    'Measles', 'BMI', 'Polio', 'Total expenditure', 'Diphtheria',
    'thinness  1-19 years', 'thinness 5-9 years',
    'Income composition of resources', 'Schooling'
]

# Function to fill missing values with the closest 3-year average
def fill_with_three_year_avg(df, column_name):
    for country in df['Country'].unique():
        country_data = df[df['Country'] == country]
        for year in country_data['Year']:
            if pd.isna(country_data.loc[country_data['Year'] == year, column_name].values[0]):
                # Get the previous, current, and next year
                prev_year = year - 1
                next_year = year + 1

                # Get the values for the previous, current, and next years
                prev_value = country_data[country_data['Year'] == prev_year][column_name]
                next_value = country_data[country_data['Year'] == next_year][column_name]

                # Initialize a list to hold valid values (non-NaN)
                valid_values = []

                if prev_value.size > 0 and not pd.isna(prev_value.values[0]):
                    valid_values.append(prev_value.values[0])

                if next_value.size > 0 and not pd.isna(next_value.values[0]):
                    valid_values.append(next_value.values[0])

                # If valid values exist, calculate the 3-year average and fill the missing value
                if valid_values:
                    df.loc[(df['Country'] == country) & (df['Year'] == year), column_name] = np.mean(valid_values)
    return df

# Apply the 3-year moving average to columns with missing data
for col in columns_with_missing_data:
    df_cleaned = fill_with_three_year_avg(df_cleaned, col)

# Now let's apply the regional average method to fill remaining missing values
def fill_with_region_avg(df, column_name):
    for status in df['Status'].unique():
        status_data = df[df['Status'] == status]
        status_avg = status_data[column_name].mean()  # Calculate the average for the region

        # Fill missing values in the specified column for countries in the status group
        df.loc[(df['Status'] == status) & (df[column_name].isna()), column_name] = status_avg
    return df

# Apply the regional average filling for missing columns
for col in columns_with_missing_data:
    df_cleaned = fill_with_region_avg(df_cleaned, col)

# Remove countries with more than 4 missing data columns
missing_data_per_country = df_cleaned.isnull().sum(axis=1)
df_cleaned = df_cleaned[missing_data_per_country <= 4]

# Check the missing data after cleaning
missing_data_after_cleaning = df_cleaned.isnull().sum()
print(missing_data_after_cleaning)
```

*Output after filling the null values:*

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

IV. Exclude the duplicated columns.

```
#Column to exclude: thinness 5-9 years as it is already included within the thinness 1-19 years
df_cleaned.drop(columns=['thinness 5-9 years', 'Total expenditure', 'infant deaths', 'Country'], inplace=True)
df_cleaned = df_cleaned.drop_duplicates()
df_cleaned['Status'] = df_cleaned['Status'].map({'Developed': 0, 'Developing': 1})

#drop total expenditure as I dont see the importance. we could just use the percentage expenditure
df_cleaned
```

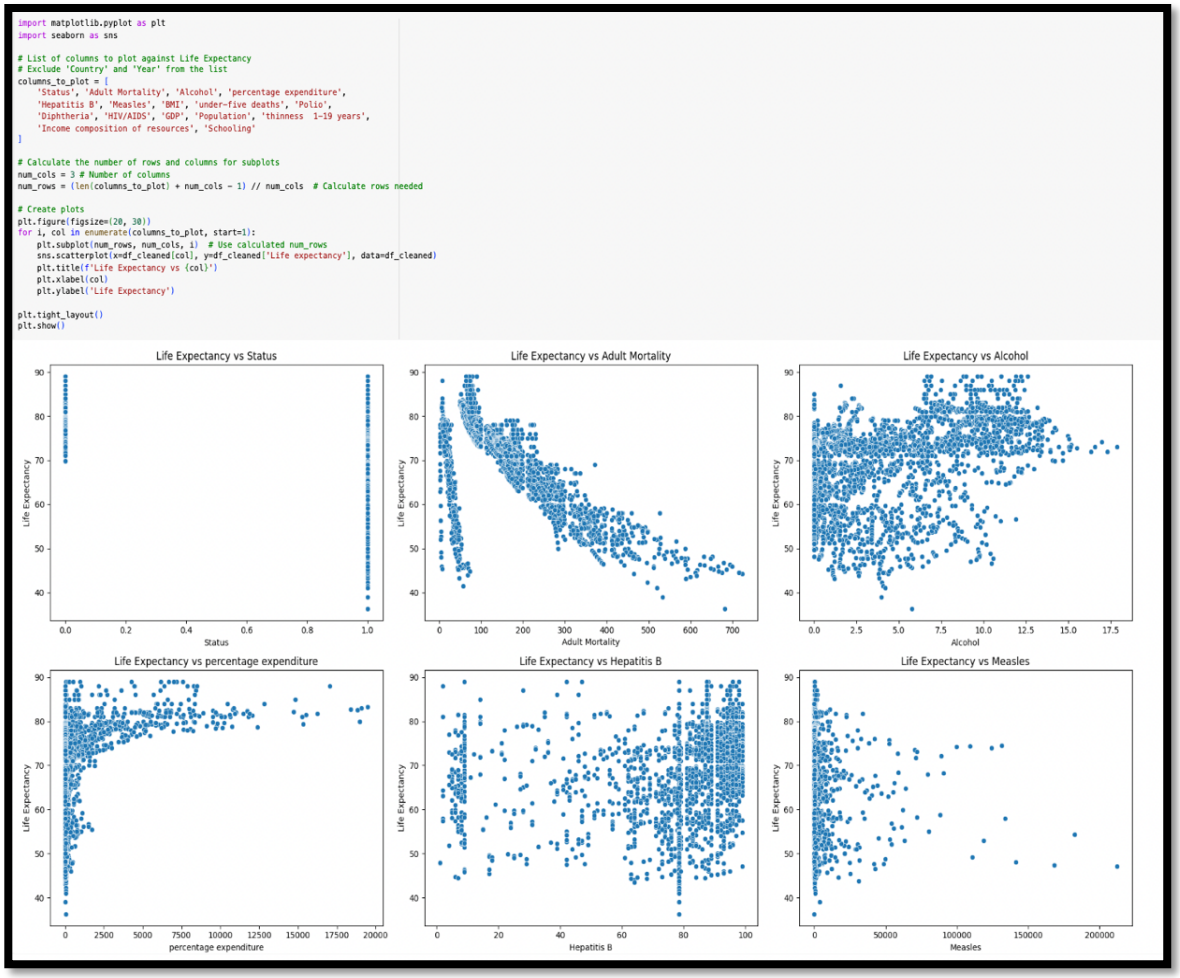
| | Year | Status | Life expectancy | Adult Mortality | Alcohol | percentage expenditure | Hepatitis B | Measles | BMI | under-five deaths | Polio | Diphtheria | HIV/AIDS | GDP | Population | thinness 1-19 years | Income composition of resources | Schooling |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2015 | 1 | 65.0 | 263.0 | 0.01 | 71.279624 | 65.0 | 1154.0 | 19.1 | 83 | 6.0 | 65.0 | 0.1 | 584.259210 | 33736494.0 | 17.2 | 0.479 | 10.1 |
| 1 | 2014 | 1 | 59.9 | 271.0 | 0.01 | 73.523582 | 62.0 | 492.0 | 18.6 | 86 | 58.0 | 62.0 | 0.1 | 612.696514 | 327582.0 | 17.5 | 0.476 | 10.0 |
| 2 | 2013 | 1 | 59.9 | 268.0 | 0.01 | 73.219243 | 64.0 | 430.0 | 18.1 | 89 | 62.0 | 64.0 | 0.1 | 631.744976 | 31731688.0 | 17.7 | 0.470 | 9.9 |
| 3 | 2012 | 1 | 59.5 | 272.0 | 0.01 | 78.184215 | 67.0 | 2787.0 | 17.6 | 93 | 67.0 | 67.0 | 0.1 | 669.959000 | 3696958.0 | 17.9 | 0.463 | 9.8 |
| 4 | 2011 | 1 | 59.2 | 275.0 | 0.01 | 7.097109 | 68.0 | 3013.0 | 17.2 | 97 | 68.0 | 68.0 | 0.1 | 63.537231 | 2978599.0 | 18.2 | 0.454 | 9.5 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2933 | 2004 | 1 | 44.3 | 723.0 | 4.36 | 0.000000 | 68.0 | 31.0 | 27.1 | 42 | 67.0 | 65.0 | 33.6 | 454.366654 | 12777511.0 | 9.4 | 0.407 | 9.2 |
| 2934 | 2003 | 1 | 44.5 | 715.0 | 4.06 | 0.000000 | 7.0 | 998.0 | 26.7 | 41 | 7.0 | 68.0 | 36.7 | 453.351155 | 12633897.0 | 9.8 | 0.418 | 9.5 |
| 2935 | 2002 | 1 | 44.8 | 73.0 | 4.43 | 0.000000 | 73.0 | 304.0 | 26.3 | 40 | 73.0 | 71.0 | 39.8 | 57.348340 | 125525.0 | 1.2 | 0.427 | 10.0 |
| 2936 | 2001 | 1 | 45.3 | 686.0 | 1.72 | 0.000000 | 76.0 | 529.0 | 25.9 | 39 | 76.0 | 75.0 | 42.1 | 548.587312 | 12366165.0 | 1.6 | 0.427 | 9.8 |
| 2937 | 2000 | 1 | 46.0 | 665.0 | 1.68 | 0.000000 | 79.0 | 1483.0 | 25.5 | 39 | 78.0 | 78.0 | 43.5 | 547.358878 | 12222251.0 | 11.0 | 0.434 | 9.8 |

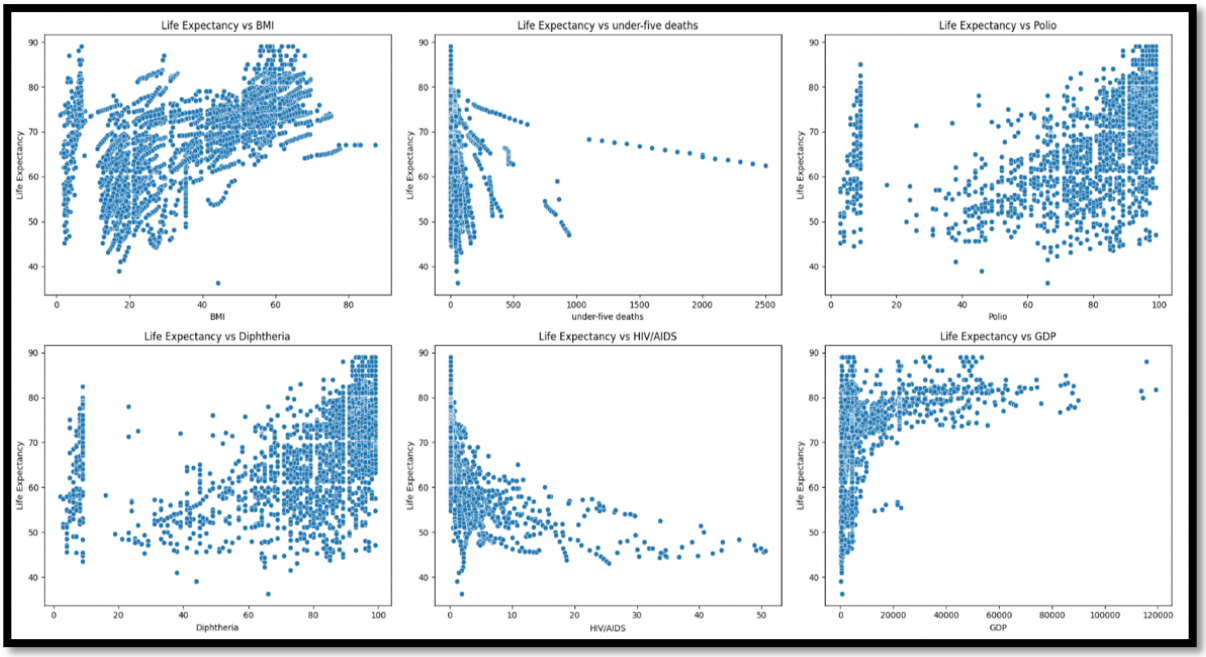
2938 rows × 18 columns

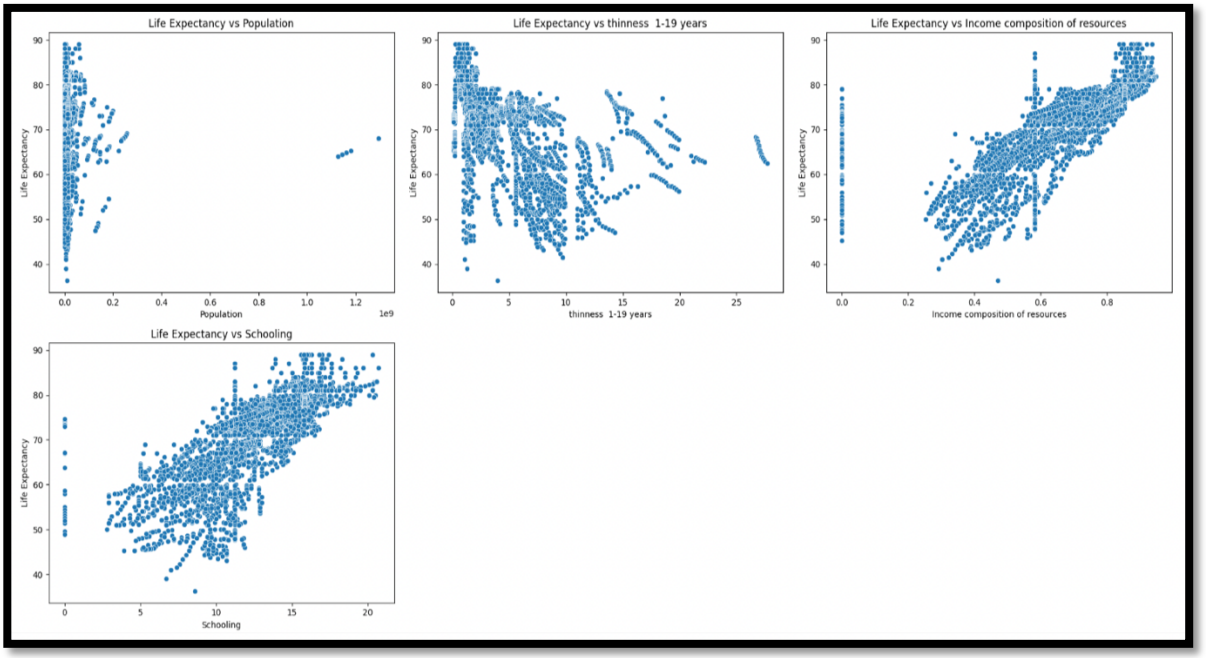
## Step 3: Explore The Dataset

I. Scatterplot to check the relationship between the target column and other columns in the dataset.

```
import matplotlib.pyplot as plt
import seaborn as sns

# List of columns to plot against Life Expectancy
# Exclude 'Country' and 'Year' from the list
columns_to_plot = [
    'Status', 'Adult Mortality', 'Alcohol', 'percentage expenditure',
    'Hepatitis B', 'Measles', 'BMI', 'under-five deaths', 'Polio',
    'Diphtheria', 'HIV/AIDS', 'GDP', 'Population', 'thinness  1-19 years',
    'Income composition of resources', 'Schooling'
]

# Calculate the number of rows and columns for subplots
num_cols = 3 # Number of columns
num_rows = (len(columns_to_plot) + num_cols - 1) // num_cols  # Calculate rows needed

# Create plots
plt.figure(figsize=(20, 30))
for i, col in enumerate(columns_to_plot, start=1):
    plt.subplot(num_rows, num_cols, i)  # Use calculated num_rows
    sns.scatterplot(x=df_cleaned[col], y=df_cleaned['Life expectancy'], data=df_cleaned)
    plt.title(f'Life Expectancy vs {col}')
    plt.xlabel(col)
    plt.ylabel('Life Expectancy')

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






*Insight:*

*a. Status: if the country is developed, the life expectancy is from 70 and above. Whereas, on the developing countries the lowest life expectancy can go lower than 40.This means the status of a country has huge effect on the life expectancy of its individuals.*

*b. Adult Mortality: As the number of adult mortality increases, the life expectancy has declined extensively as shown on the scatter plot. This means, adult mortality holds a huge weight on the life expectancy of a country.*

*c. Measles: When the number of measles were close to 0, the life expectancy came close to 90, but as the number of measles increases, the life expectancy highest hit was no more than 50 to 60 years.*

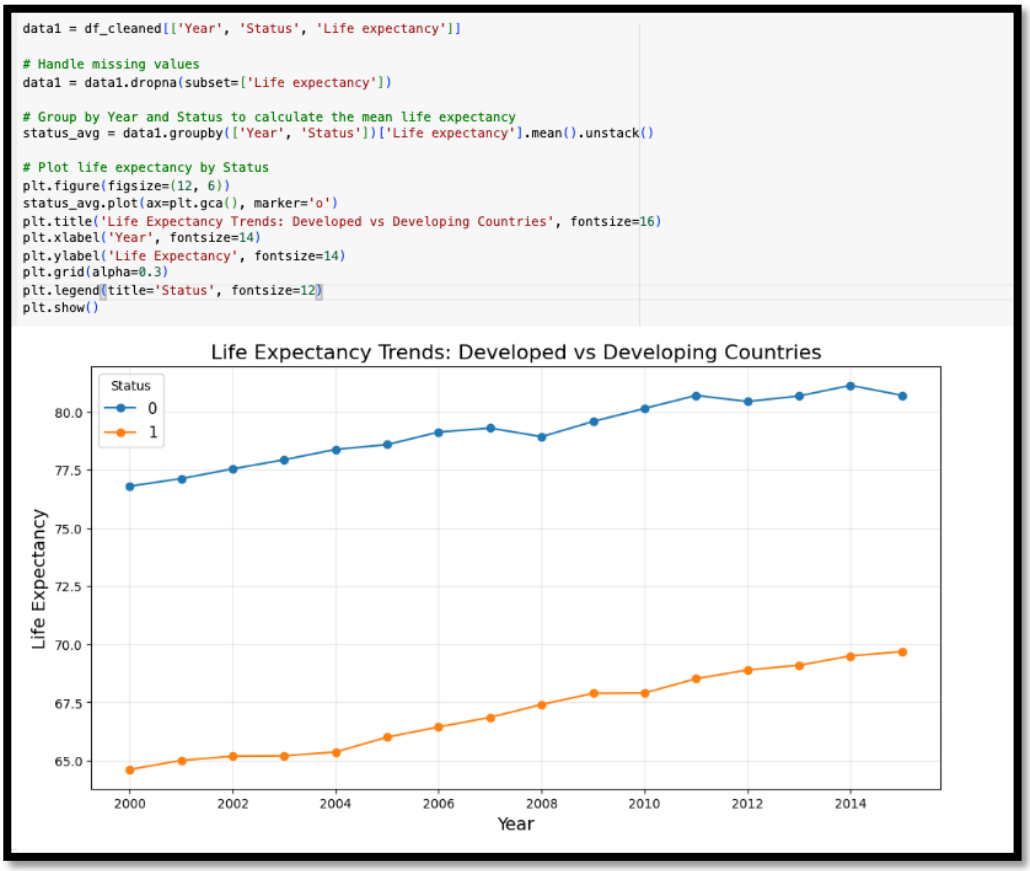
*d. Schooling and Income composition of resources: These two factors affect the life expectancy of a country significantly, as the scatter plot shows the life expectancy increases along with the schooling and income composition of resources.*

*e. GDP:when the Gross domestic product (GDP) increases, the life expectncy of the country increases.*

*f. thinness 1- 19 years: as the thinness increases, the probablity of an individual living longer decreases which is clearly seen on the plot.*

II. Check the progress of life expectancy over the years in developing and developed countries.

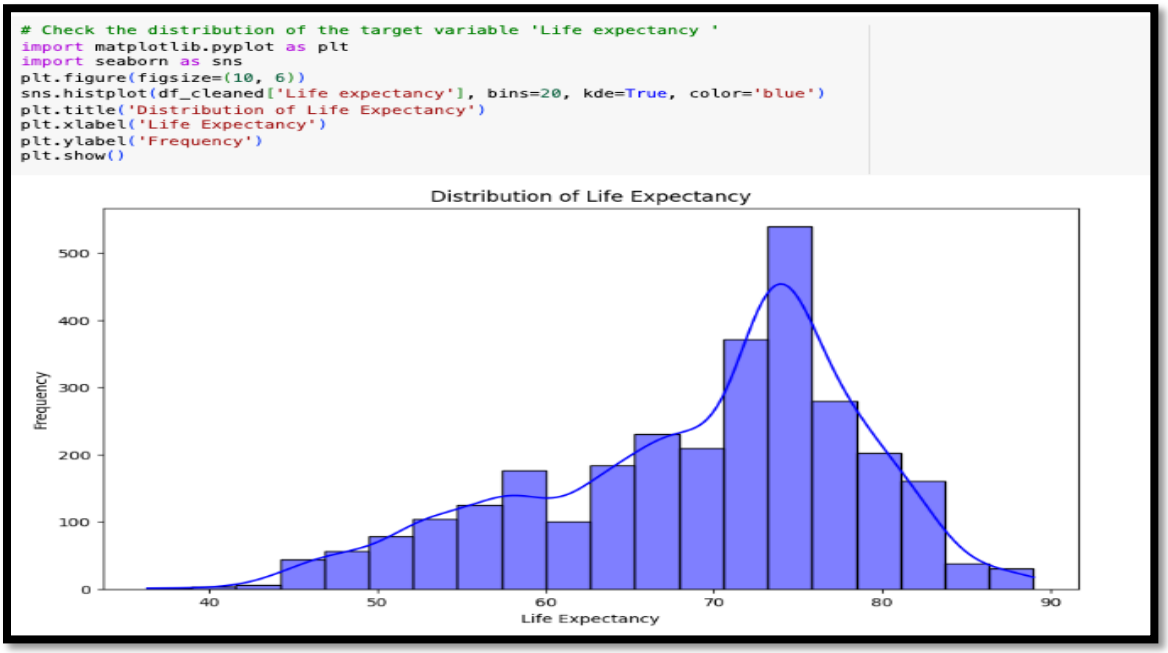
```
data1 = df_cleaned[['Year', 'Status', 'Life expectancy']]

# Handle missing values
data1 = data1.dropna(subset=['Life expectancy'])

# Group by Year and Status to calculate the mean life expectancy
status_avg = data1.groupby(['Year', 'Status'])['Life expectancy'].mean().unstack()

# Plot life expectancy by Status
plt.figure(figsize=(12, 6))
status_avg.plot(ax=plt.gca(), marker='o')
plt.title('Life Expectancy Trends: Developed vs Developing Countries', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Life Expectancy', fontsize=14)
plt.grid(alpha=0.3)
plt.legend(title='Status', fontsize=12)
plt.show()
```



***Insight:*** *We encoded the categorical column "Status" by assigning the value 0 to developed countries and 1 to developing countries. The analysis revealed that life expectancy has been increasing over the years. However, developed countries consistently show life expectancy levels of 75 years and above, while in developing countries, life expectancy has improved from 65 to approximately 70 years. This significant gap highlights how factors such as infrastructure, living standards, and economic conditions profoundly influence life expectancy.*

III. Check the distribution of the life expectancy.

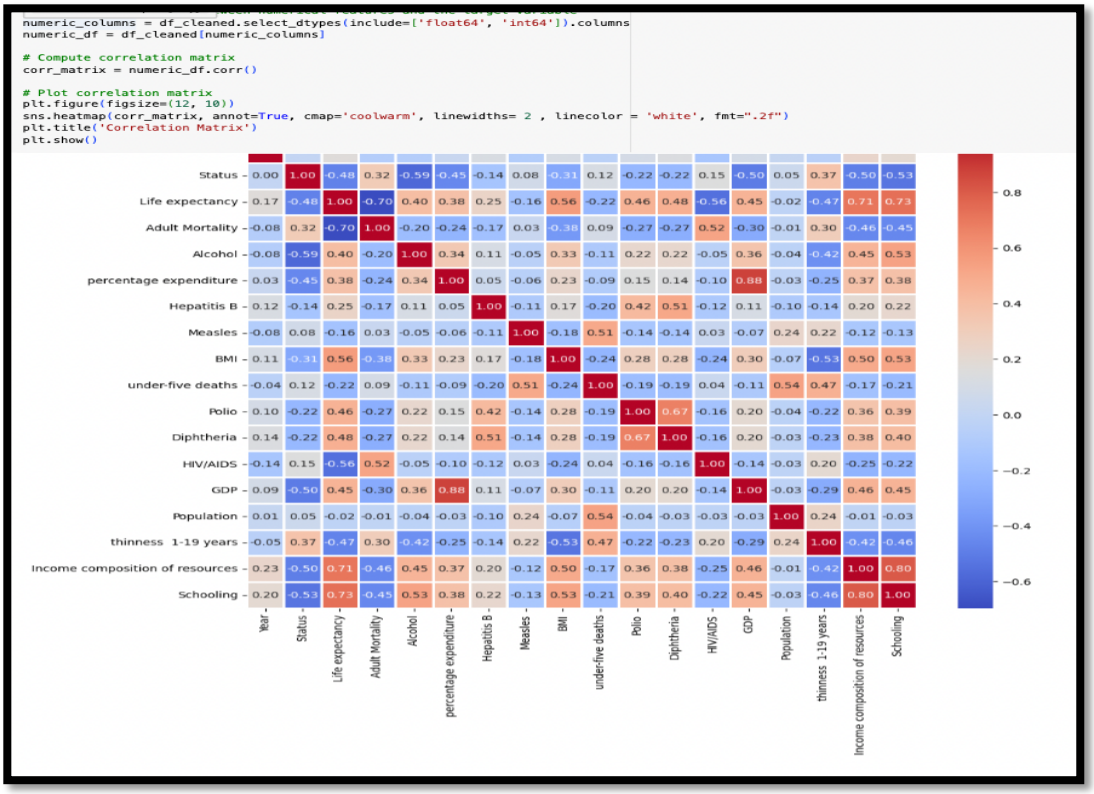
```
# Check the distribution of the target variable 'Life expectancy '
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.histplot(df_cleaned['Life expectancy'], bins=20, kde=True, color='blue')
plt.title('Distribution of Life Expectancy')
plt.xlabel('Life Expectancy')
plt.ylabel('Frequency')
plt.show()
```




***Insight:*** *The results indicate that a life expectancy of 75 years is the most common across all countries.*

IV. Create a heatmap to see any correlation between the columns.

```
numeric_columns = df_cleaned.select_dtypes(include=['float64', 'int64']).columns
numeric_df = df_cleaned[numeric_columns]

# Compute correlation matrix
corr_matrix = numeric_df.corr()

# Plot correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths= 2 , linecolor = 'white', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```




***Insight:*** *The income composition of resources has a correlation of 0.71 with life expectancy, while schooling has a correlation of 0.73 with life expectancy. This indicates that higher levels of schooling and income composition are directly proportional to longer life expectancy. Conversely, adult mortality has a correlation of -0.7 with life expectancy, signifying an inverse relationship where an increase in adult mortality leads to a decrease in a country's life expectancy.*

## Step 4: Feature Selection

For feature selection, we used mutual information, which quantifies how much knowledge of one feature increases the confidence or probability of accurately predicting life expectancy.

```
df_cleaned.columns
```

```
Index(['Year', 'Status', 'Life expectancy', 'Adult Mortality', 'Alcohol',
       'percentage expenditure', 'Hepatitis B', 'Measles', 'BMI',
       'under-five deaths', 'Polio', 'Diphtheria', 'HIV/AIDS', 'GDP',
       'Population', 'thinness  1-19 years', 'Income composition of resources',
       'Schooling'],
      dtype='object')
```

```
from sklearn.feature_selection import mutual_info_regression
import pandas as pd

df1=df_cleaned.drop(columns=['Year'])
# Separate features and target variable
# Use the cleaned DataFrame (df_cleaned) for both X and y
X = df1.drop(columns=['Life expectancy'])
y = df1['Life expectancy']

# Calculate mutual information
mutual_info = mutual_info_regression(X, y)

# Create a DataFrame to show features and their importance
mutual_info_df = pd.DataFrame({'Feature': X.columns, 'Mutual Information': mutual_info})

# Sort by mutual information
mutual_info_df = mutual_info_df.sort_values(by='Mutual Information', ascending=False)

print(mutual_info_df)
```

```
                           Feature  Mutual Information
1                   Adult Mortality            1.283750
14  Income composition of resources            0.943343
13              thinness  1-19 years            0.791114
15                        Schooling            0.702564
6                               BMI            0.578915
10                         HIV/AIDS            0.541435
7                 under-five deaths            0.434211
2                           Alcohol            0.369056
11                              GDP            0.368116
8                             Polio            0.332881
9                        Diphtheria            0.302938
3            percentage expenditure            0.288013
4                       Hepatitis B            0.277509
0                            Status            0.207801
12                       Population            0.163519
5                           Measles            0.120499
```

***Insight***: *The highest mutual information is 1.283750, indicating that knowing the adult mortality significantly improves the probability of accurately predicting life expectancy. This trend is similar to other columns with high mutual information values, suggesting they also contribute meaningfully to predicting life expectancy.* In the end, we selected the top 8 features with the highest mutual information values and excluded the bottom 8 features, as their significance diminishes further down the list.

## Step 5: Model Creation

I. To create the model, we used the selected features for better results.

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import matplotlib.pyplot as plt


# 1. Select the top 8 features based on mutual information scores
selected_features = [
    'Adult Mortality',
    'Income composition of resources',
    'thinness  1-19 years',
    'Schooling',
    'BMI',
    'HIV/AIDS',
    'under-five deaths',
    'GDP'
]

# Drop rows with missing values in the target variable 'Life expectancy'
df_cleaned = df_cleaned.dropna(subset=['Life expectancy'])

# Select the features (X) and target (y)
X = df_cleaned[selected_features]  # Features
y = df_cleaned['Life expectancy']  # Target

# 3. Scale the features using StandardScaler (standardize the data)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

II. We defined a function that will look through all the possible models and choose the best model based on the results of the evaluation metrics.

```python
from sklearn.model_selection import train_test_split, cross_val_score
from (module) sklearn import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
import numpy as np
import pandas as pd

def compare_models(X, y):
    # Define the models to compare
    models = {
        'GradientBoostingRegressor': GradientBoostingRegressor(),
        'LinearRegression': LinearRegression(),
        'RandomForestRegressor': RandomForestRegressor(),
        'XGBRegressor': XGBRegressor(eval_metric='rmse'),
        'DecisionTreeRegressor': DecisionTreeRegressor(),
        'SVR': SVR()
    }

    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Store the results
    results = []

    # Evaluate each model
    for model_name, model in models.items():
        model.fit(X_train, y_train)  # Train the model
        y_pred = model.predict(X_test)  # Predict on the test set

        # Calculate metrics
        mae = mean_absolute_error(y_test, y_pred)
        rmse = np.sqrt(mean_squared_error(y_test, y_pred))
        cv_rmse = -np.mean(cross_val_score(model, X, y, scoring='neg_root_mean_squared_error', cv=5))
        r2 = r2_score(y_test, y_pred)

        # Append results
        results.append({
            'Model': model_name,
            'MAE': mae,
            'RMSE': rmse,
            'CV-RMSE': cv_rmse,
            'R²': r2
        })

    # Convert results to a DataFrame
    results_df = pd.DataFrame(results)

    # Determine the best model for each metric
    best_models = {
        'Best MAE': results_df.loc[results_df['MAE'].idxmin()]['Model'],
        'Best RMSE': results_df.loc[results_df['RMSE'].idxmin()]['Model'],
        'Best CV-RMSE': results_df.loc[results_df['CV-RMSE'].idxmin()]['Model'],
        'Best R²': results_df.loc[results_df['R²'].idxmax()]['Model']
    }

    return results_df, best_models


results, best_models = compare_models(X, y)
print(results)
print(best_models)
```

```
                        Model       MAE      RMSE   CV-RMSE        R²
0  GradientBoostingRegressor  1.591314  2.193781  2.863051  0.944457
1           LinearRegression  3.073920  4.278392  4.668643  0.788746
2      RandomForestRegressor  1.067817  1.675114  2.851160  0.967616
3              XGBRegressor  1.154643  1.771269  2.985875  0.963791
4      DecisionTreeRegressor  1.556254  2.639000  3.971691  0.919625
5                        SVR  5.642254  7.574186  7.897013  0.337911
{'Best MAE': 'RandomForestRegressor', 'Best RMSE': 'RandomForestRegressor', 'Best CV-RMSE': 'RandomForestRegressor', 'Best R²': 'RandomForestRegressor'}
```

*The results indicated that the RandomForestRegressor outperformed other algorithms, making it the most effective choice based on the evaluation metrics provided.*

III. After selecting the RandomForestRegressor, we proceeded to create and train our model based on the selected features. The model's performance was tested on the predicted outputs, which yielded promising results, making it a reliable tool for predicting future life expectancy.

```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import matplotlib.pyplot as plt


# 1. Select the top 8 features based on mutual information scores
selected_features = [
    'Adult Mortality',
    'Income composition of resources',
    'thinness  1-19 years',
    'Schooling',
    'BMI',
    'HIV/AIDS',
    'under-five deaths',
    'GDP'
]

# Drop rows with missing values in the target variable 'Life expectancy'
df_cleaned = df_cleaned.dropna(subset=['Life expectancy'])

# Select the features (X) and target (y)
X = df_cleaned[selected_features]  # Features
y = df_cleaned['Life expectancy']  # Target

# 3. Scale the features using StandardScaler (standardize the data)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# 5. Initialize a regression model (Random Forest Regressor)
model = RandomForestRegressor(n_estimators=100, random_state=42)

# 6. Train the model
model.fit(X_train, y_train)

# 7. Make predictions on the test set
y_pred = model.predict(X_test)

# 8. Calculate the necessary metrics
# 8.1 Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)

# 8.2 Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# 8.3 Cross-validation score (CV-SCORE)
cv_scores = cross_val_score(model, X_scaled, y, cv=5, scoring='neg_mean_squared_error')  # Negative MSE is returned
cv_score = np.mean(np.sqrt(-cv_scores))  # Converting back from negative MSE to RMSE

# 8.4 R-squared (R²)
r2 = r2_score(y_test, y_pred)

# 9. Print the evaluation metrics
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"Cross-Validation RMSE (CV-SCORE): {cv_score:.4f}")
print(f"R-squared (R²): {r2:.4f}")
```

```
Mean Absolute Error (MAE): 1.0729
Root Mean Squared Error (RMSE): 1.6908
Cross-Validation RMSE (CV-SCORE): 2.8773
R-squared (R²): 0.9670
```

## References


Abhinaya, V., Dharani, C. B., Vandana, A., Velvadivu, P., & Sathya, C. (n.d.). *Statistical analysis on factors influencing ...* International Research Journal of Engineering and Technology (IRJET). https://www.irjet.net/archives/V8/i7/IRJET-V8I7785.pdf

KumarRajarshi. (2018, February 10). *Life expectancy (WHO)*. Kaggle. https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who/code

Ronmi, A. E., Prasad, R., & Raphael , B. A. (2023, March 7). *How can artificial intelligence and data science algorithms predict life expectancy - an empirical investigation spanning 193 countries*. International Journal of Information Management Data Insights. https://www.sciencedirect.com/science/article/pii/S2667096823000150

Roy, K., & Swargiary, K. (2024). GLOBAL LIFE EXPECTANCY TRENDS: INFLUENCING FACTORS, DISPARITIES, AND POLICY IMPLICATIONS IN 2024. https://www.researchgate.net/publication/378240798_GLOBAL_LIFE_EXPECTANCY_TRENDS_INFLUENCING_FACTORS_DISPARITIES_AND_POLICY_IMPLICATIONS_IN_2024


# Group Members


Rahel Tekle

Saron Haile

Snit Daniel