STEP 1: 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()
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/kumarajarshi/life-expectancy-who?dataset_version_number
100%|██████████| 119k/119k [00:00<00:00, 36.1MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/kumarajarshi/life-expectancy-who/versions/1
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Tot expenditu
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7

5 rows × 22 columns

```
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
```

```
df.columns
```

```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

```
df.describe()
```

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	unde fi deat
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.000000	2938.000
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.321247	42.035
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.044034	160.445
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.300000	0.000
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	43.500000	4.000
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	56.200000	28.000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000

```
df.shape
```

```
(2938, 22)
```

STEP 2: CLEAN THE DATASET

```
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
```

```
# 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
```

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.

```
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)
```

```
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
<ipython-input-8-e6781f4f4555>:52: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an
  df.loc[(df['Status'] == status) & (df[column_name].isna()), column_name] = status_avg
```

```
#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
```

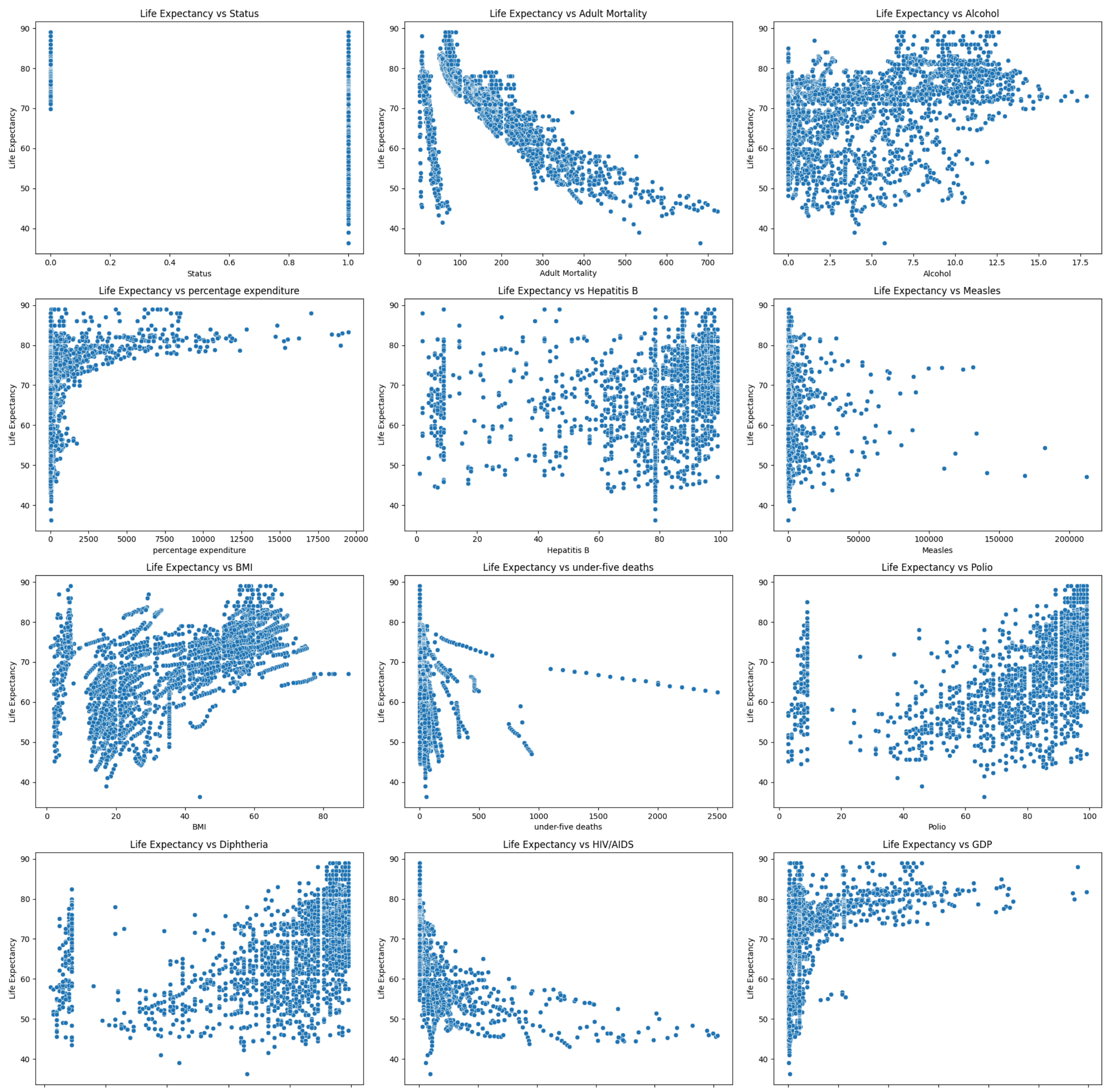
	Year	Status	Life expectancy	Adult Mortality	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Diphtheria	HIV/AI
0	2015	1	65.0	263.0	0.01	71.279624	65.0	1154.0	19.1	83	6.0	65.0	0
1	2014	1	59.9	271.0	0.01	73.523582	62.0	492.0	18.6	86	58.0	62.0	0
2	2013	1	59.9	268.0	0.01	73.219243	64.0	430.0	18.1	89	62.0	64.0	0
3	2012	1	59.5	272.0	0.01	78.184215	67.0	2787.0	17.6	93	67.0	67.0	0
4	2011	1	59.2	275.0	0.01	7.097109	68.0	3013.0	17.2	97	68.0	68.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	
2933	2004	1	44.3	723.0	4.36	0.000000	68.0	31.0	27.1	42	67.0	65.0	33
2934	2003	1	44.5	715.0	4.06	0.000000	7.0	998.0	26.7	41	7.0	68.0	36
2935	2002	1	44.8	73.0	4.43	0.000000	73.0	304.0	26.3	40	73.0	71.0	39
2936	2001	1	45.3	686.0	1.72	0.000000	76.0	529.0	25.9	39	76.0	75.0	42
2937	2000	1	46.0	665.0	1.68	0.000000	79.0	1483.0	25.5	39	78.0	78.0	43

STEP 3: EXPLORE THE DATASET

```
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')
```

```
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()
```



Based on the above plots the most impactful columns are: 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.

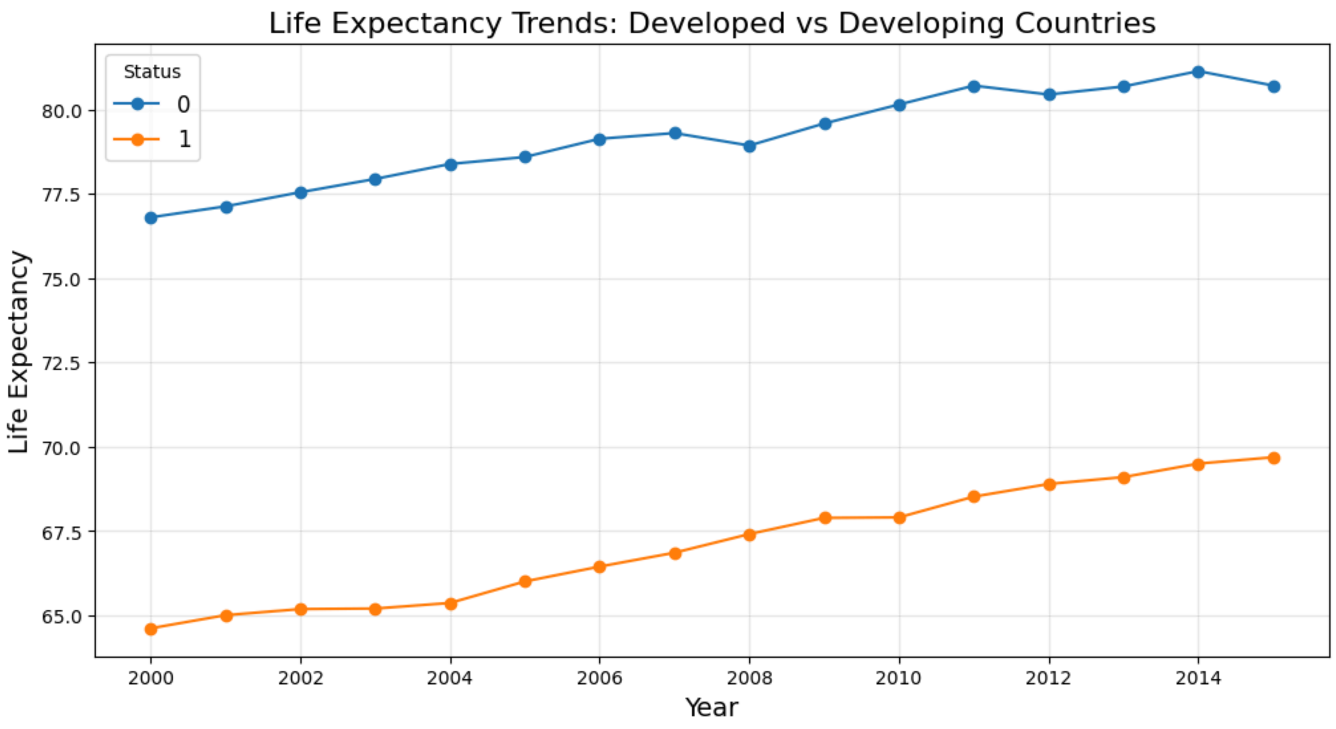
d. Schooling and Income composition of resources: These two factors affect the life expectancy of a country significantly, as the scatter plot shows the life expectancy increases along with the schooling and income composition of resources.

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

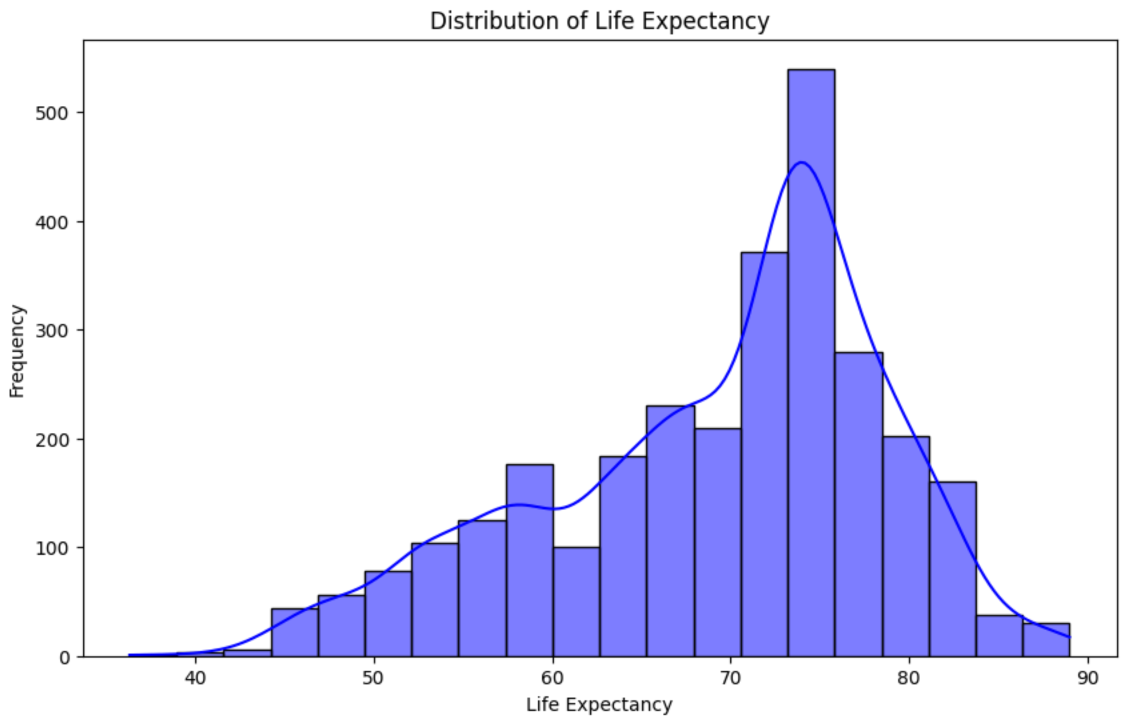
f. thinness 1- 19 years: as the thinness increases, the probablity of an individual living longer decreases which is clearly seen on the plot.

```
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()
```

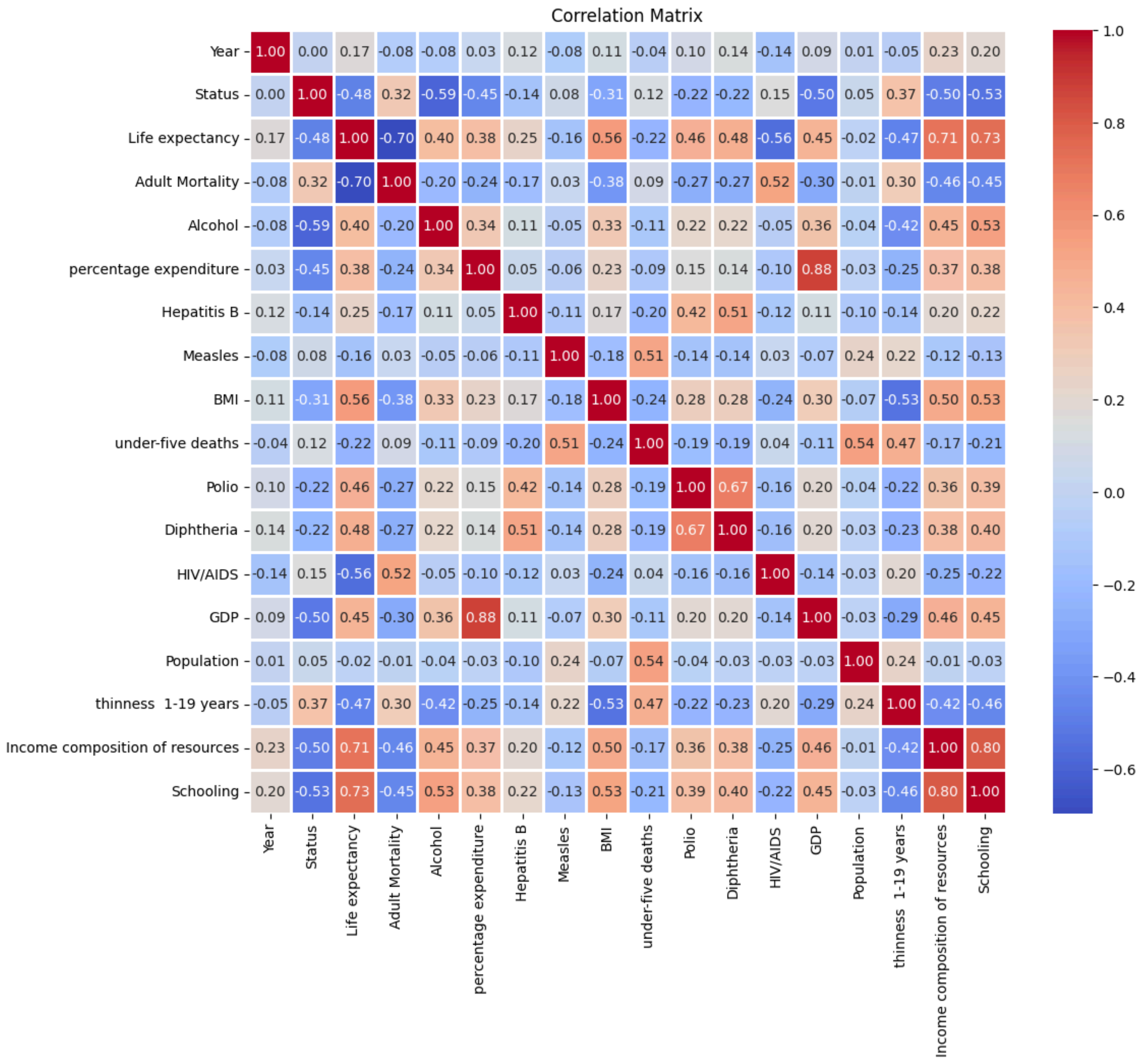


```
# Check the distribution of the target variable 'Life expectancy '
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.histplot(df_cleaned['Life expectancy'], bins=20, kde=True, color='blue')
plt.title('Distribution of Life Expectancy')
plt.xlabel('Life Expectancy')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



```
# Explore correlations between numerical features and the target variable
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()
```



STEP 4: FEATURE SELECTION

```
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
```

STEP 5: Model Creation

```
# 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)


from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics 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', 'Be
```

Based on the evaluation metrics, the RandomForest Regressor has outperformed the other models.

```
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
```