Importing necessary libraries

```
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For data visualization
import seaborn as sns  # For advanced data visualization
from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
from sklearn.preprocessing import StandardScaler  # For feature scaling
from sklearn.linear_model import LinearRegression  # Linear Regression algorithm
from sklearn.ensemble import RandomForestRegressor  # Random Forest algorithm
from xgboost import XGBRegressor  # XGBoost algorithm
from sklearn.svm import SVR  # Support Vector Regression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score  # Evaluation metrics
from sklearn.impute import SimpleImputer  # For handling missing values
```

Load the dataset

Assuming the dataset is downloaded and saved as 'Life Expectancy Data.csv'

```
df = pd.read_excel('led.xlsx')  # Load the dataset
```

Display basic information about the dataset

```
print("Dataset Information:")
print(df.info())  # Shows columns, non-null counts, and data types
print("\nFirst 5 rows of the dataset:")
print(df.head())  # Displays first 5 rows to understand the data structure
```

```
 5   infantdeaths                 2938 non-null   int64
 6   Alcohol                      2744 non-null   float64
 7   percentageexpenditure        2938 non-null   float64
 8   HepatitisB                   2385 non-null   float64
 9   Measles                      2938 non-null   int64
 10  BMI                          2904 non-null   float64
 11  under-fivedeaths             2938 non-null   int64
 12  Polio                        2919 non-null   float64
 13  Totalexpenditure             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  thinness1-19years            2904 non-null   float64
 19  thinness5-9years             2904 non-null   float64
 20  Incomecompositionofresources  2771 non-null   float64
 21  Schooling                    2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
None

First 5 rows of the dataset:
       Country  Year      Status  Lifeexpectancy  AdultMortality  \
0  Afghanistan  2015  Developing            65.0           263.0
1  Afghanistan  2014  Developing            59.9           271.0
2  Afghanistan  2013  Developing            59.9           268.0
3  Afghanistan  2012  Developing            59.5           272.0
4  Afghanistan  2011  Developing            59.2           275.0

   infantdeaths  Alcohol  percentageexpenditure  HepatitisB  Measles  ...  \
0            62     0.01              71.279624        65.0     1154  ...
```

```
   thinness1-19years  thinness5-9years  Incomecompositionofresources  \
0               17.2              17.3                         0.479
1               17.5              17.5                         0.476
2               17.7              17.7                         0.470
3               17.9              18.0                         0.463
4               18.2              18.2                         0.454

   Schooling
0       10.1
1       10.0
2        9.9
3        9.8
4        9.5

[5 rows x 22 columns]
```

```
df.head()
```

	Country	Year	Status	Lifeexpectancy	AdultMortality	infantdeaths	Alcohol	percentageexpenditure	HepatitisB	Measle
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	115
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	49
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	43
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	278
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	301

5 rows × 22 columns

Data Preprocessing

Check for missing values

```
print("\nMissing values in each column:")
print(df.isnull().sum())  # Counts null values in each column
```

```
Missing values in each column:
Country                           0
Year                              0
Status                            0
Lifeexpectancy                   10
AdultMortality                   10
infantdeaths                      0
Alcohol                         194
percentageexpenditure             0
HepatitisB                      553
Measles                           0
BMI                              34
under-fivedeaths                  0
Polio                            19
Totalexpenditure                226
Diphtheria                       19
HIV/AIDS                          0
GDP                             448
Population                      652
thinness1-19years                34
thinness5-9years                 34
Incomecompositionofresources    167
Schooling                       163
dtype: int64
```

Handle missing values

We'll use median imputation for numerical columns and mode for categorical

```
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns
```

Create imputers

```
num_imputer = SimpleImputer(strategy='median')  # Replaces missing values with median
cat_imputer = SimpleImputer(strategy='most_frequent')  # Replaces missing values with most frequent value
```

Apply imputation

```
df[numerical_cols] = num_imputer.fit_transform(df[numerical_cols])
df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
```

Verify no missing values remain

```
print("\nMissing values after imputation:")
print(df.isnull().sum())
```

```
Missing values after imputation:
Country                        0
Year                           0
Status                         0
Lifeexpectancy                 0
AdultMortality                 0
infantdeaths                   0
Alcohol                        0
percentageexpenditure          0
HepatitisB                     0
Measles                        0
BMI                            0
under-fivedeaths               0
Polio                          0
Totalexpenditure               0
Diphtheria                     0
HIV/AIDS                       0
GDP                            0
Population                     0
thinness1-19years              0
thinness5-9years               0
Incomecompositionofresources   0
Schooling                      0
dtype: int64
```

Exploratory Data Analysis (EDA)

Summary statistics

```
print("\nSummary statistics:")
print(df.describe())  # Shows count, mean, std, min, max, etc.
```

```
Summary statistics:
              Year  Lifeexpectancy  AdultMortality  infantdeaths      Alcohol  \
count  2938.000000     2938.000000     2938.000000   2938.000000  2938.000000
mean   2007.518720       69.234717      164.725664     30.303948     4.546875
std       4.613841        9.509115      124.086215    117.926501     3.921946
min    2000.000000       36.300000        1.000000      0.000000     0.010000
25%    2004.000000       63.200000       74.000000      0.000000     1.092500
50%    2008.000000       72.100000      144.000000      3.000000     3.755000
75%    2012.000000       75.600000      227.000000     22.000000     7.390000
max    2015.000000       89.000000      723.000000   1800.000000    17.870000

       percentageexpenditure   HepatitisB        Measles          BMI  \
count            2938.000000  2938.000000    2938.000000  2938.000000
mean              738.251295    83.022124    2419.592240    38.381178
std              1987.914858    22.996984   11467.272489    19.935375
min                 0.000000     1.000000       0.000000     1.000000
25%                 4.685343    82.000000       0.000000    19.400000
50%                64.912906    92.000000      17.000000    43.500000
75%               441.534144    96.000000     360.250000    56.100000
max             19479.911610    99.000000  212183.000000    87.300000
```

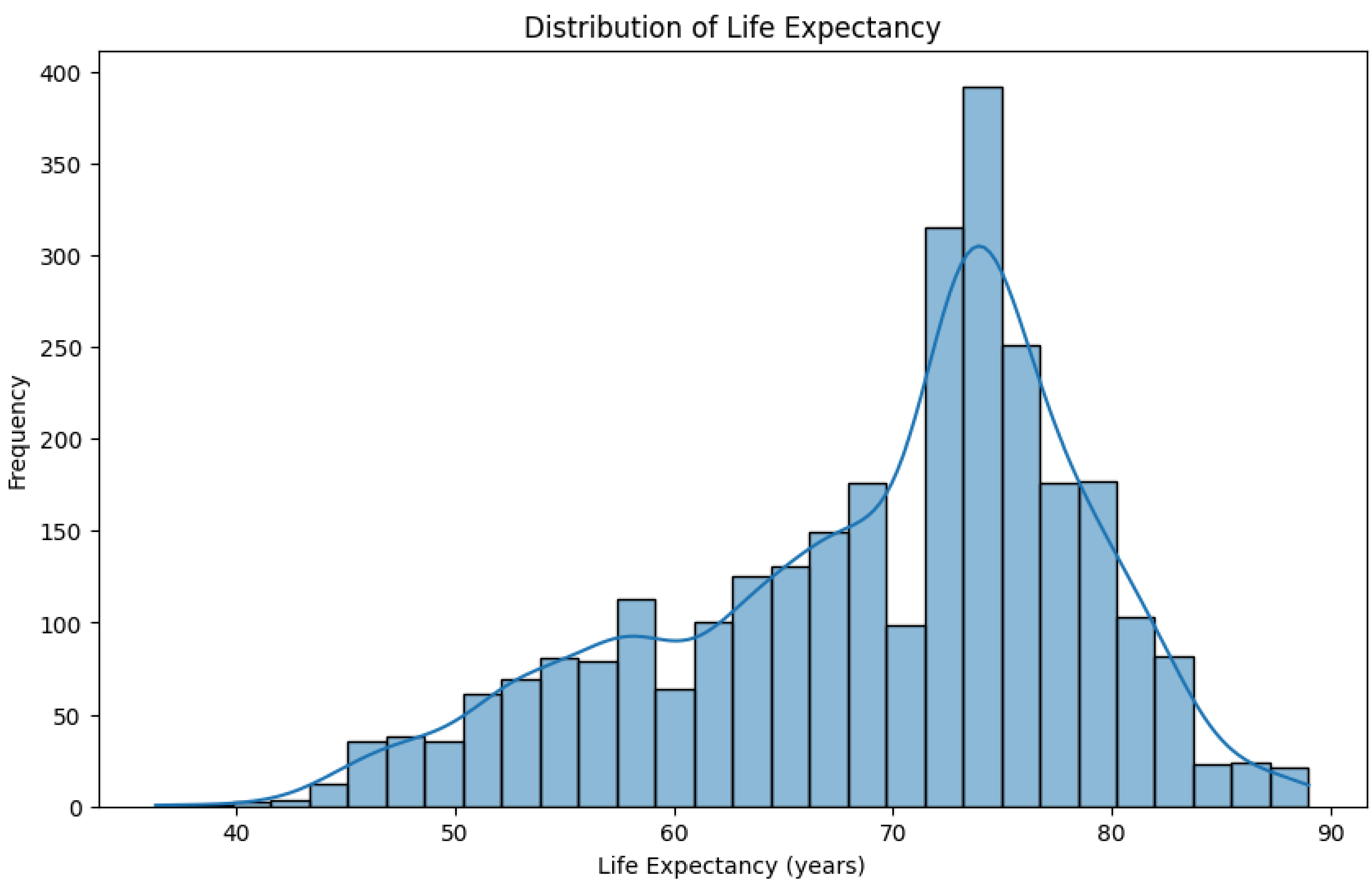
```
       under-fivedeaths        Polio  Totalexpenditure   Diphtheria  \
count       2938.000000  2938.000000       2938.000000  2938.000000
mean          42.035739    82.617767          5.924098    82.393125
std          160.445548    23.367166          2.400770    23.655562
min            0.000000     3.000000          0.370000     2.000000
25%            0.000000    78.000000          4.370000    78.000000
50%            4.000000    93.000000          5.755000    93.000000
75%           28.000000    97.000000          7.330000    97.000000
max         2500.000000    99.000000         17.600000    99.000000

          HIV/AIDS            GDP    Population  thinness1-19years  \
count  2938.000000    2938.000000  2.938000e+03        2938.000000
mean      1.742103    6611.523863  1.023085e+07           4.821886
std       5.077785   13296.603449  5.402242e+07           4.397621
min       0.100000       1.681350  3.400000e+01           0.100000
25%       0.100000     580.486996  4.189172e+05           1.600000
50%       0.100000    1766.947595  1.386542e+06           3.300000
75%       0.800000    4779.405190  4.584371e+06           7.100000
max      50.600000  119172.741800  1.293859e+09          27.700000

       thinness5-9years  Incomecompositionofresources    Schooling
count       2938.000000                   2938.000000  2938.000000
mean           4.852144                      0.630362    12.009837
std            4.485854                      0.205140     3.265139
min            0.100000                      0.000000     0.000000
25%            1.600000                      0.504250    10.300000
50%            3.300000                      0.677000    12.300000
75%            7.200000                      0.772000    14.100000
max           28.600000                      0.948000    20.700000
```

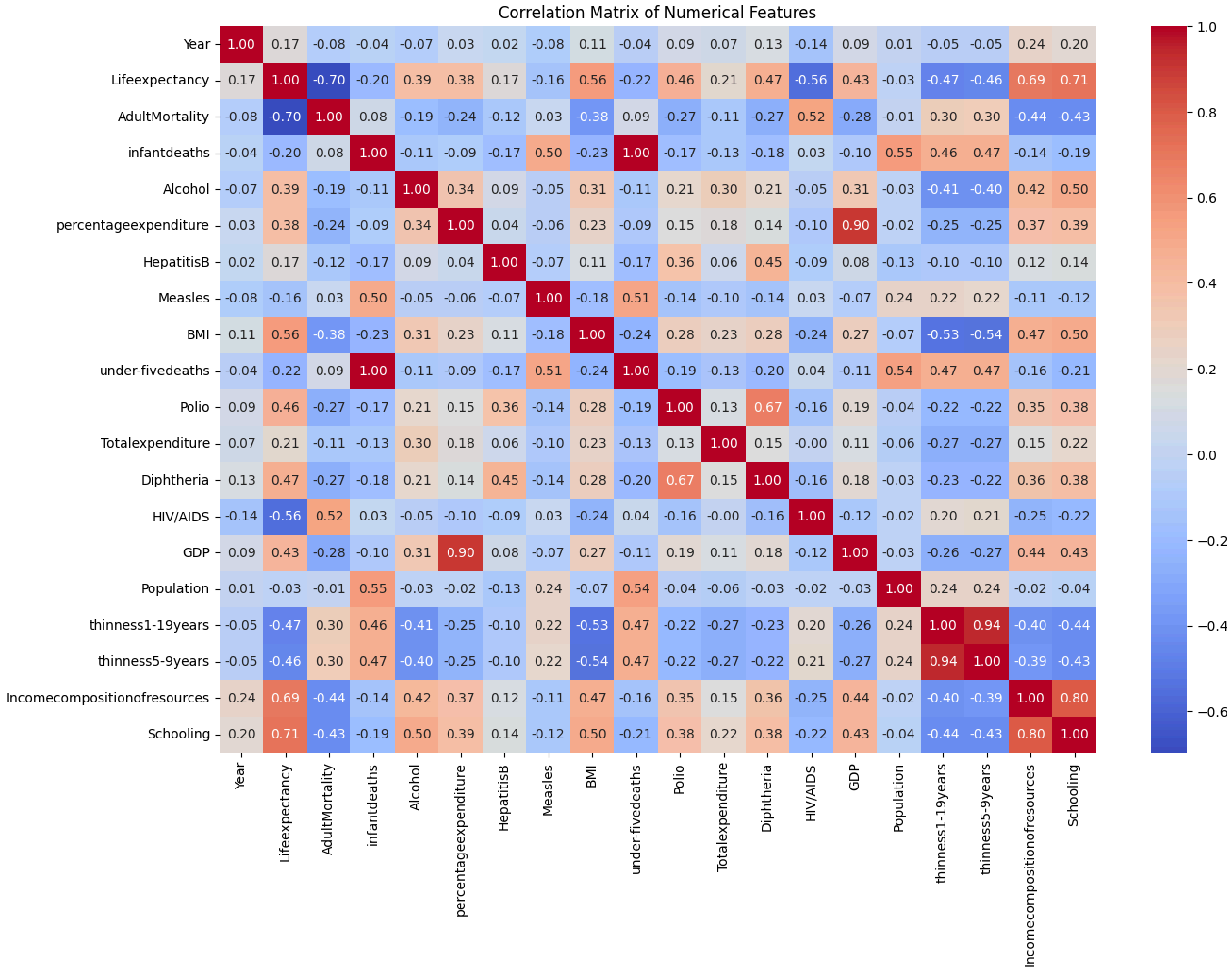
Visualize the distribution of the target variable (Life expectancy)

```
plt.figure(figsize=(10, 6))
sns.histplot(df['Lifeexpectancy'],kde=True, bins=30)
plt.title('Distribution of Life Expectancy')
plt.xlabel('Life Expectancy (years)')
plt.ylabel('Frequency')
plt.show()
```

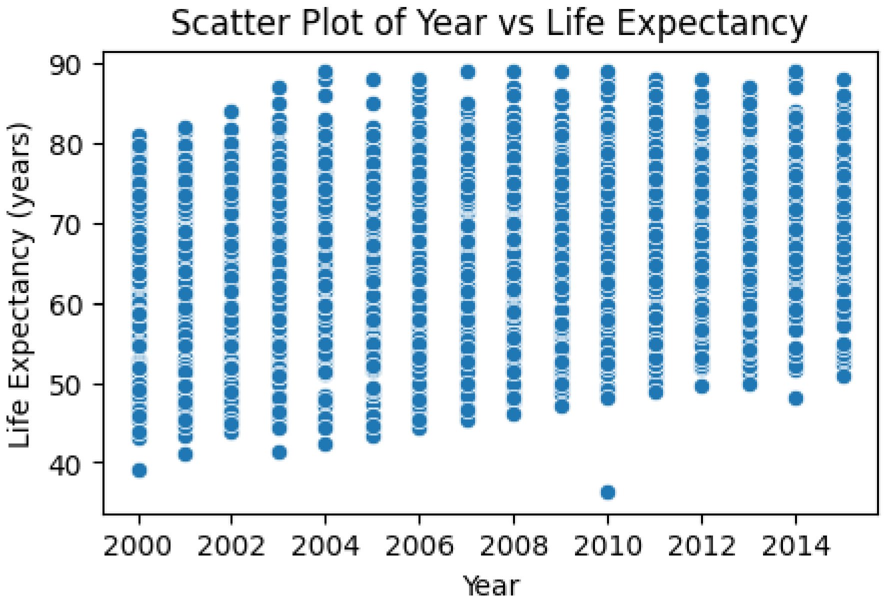


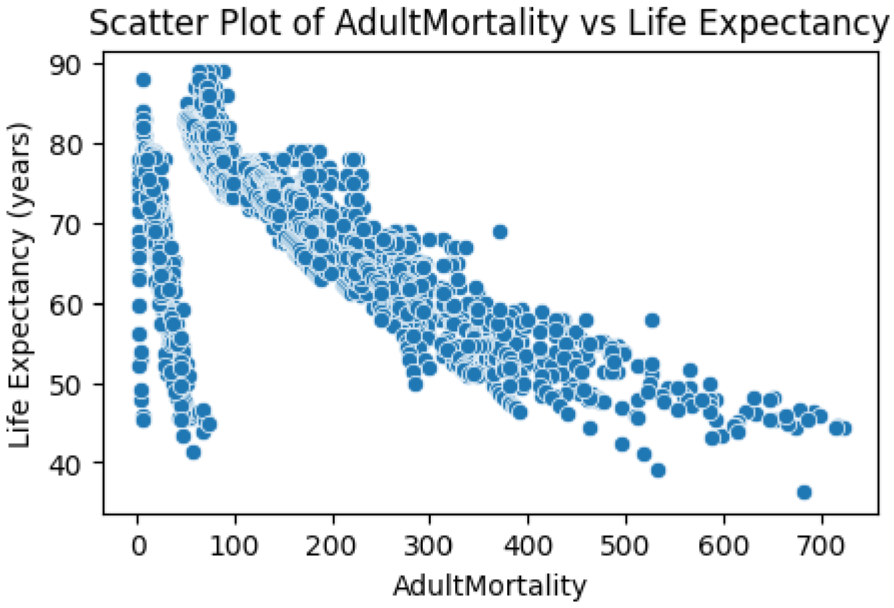
Correlation matrix to understand relationships between variables

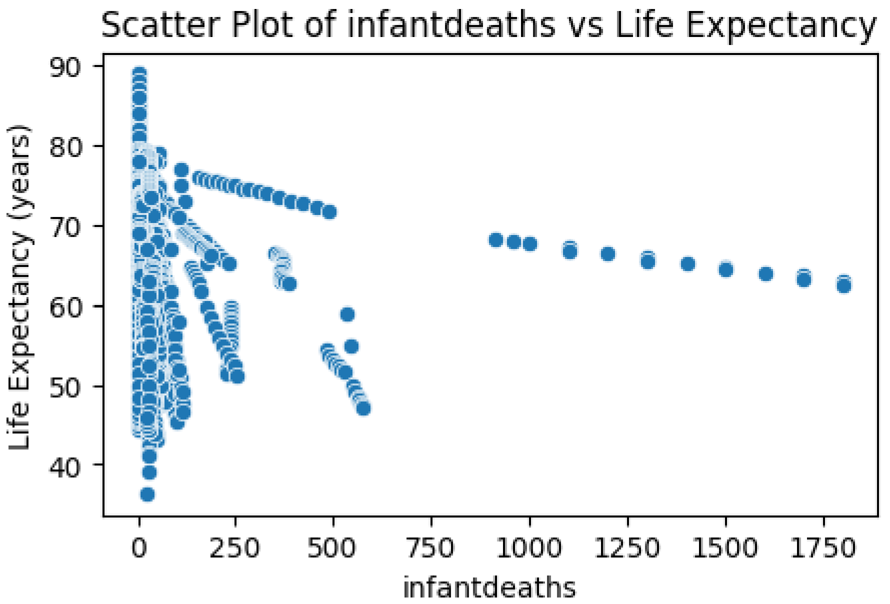
```
plt.figure(figsize=(15, 10))
corr_matrix = df.corr(numeric_only=True)  # Calculates correlation between numerical columns
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

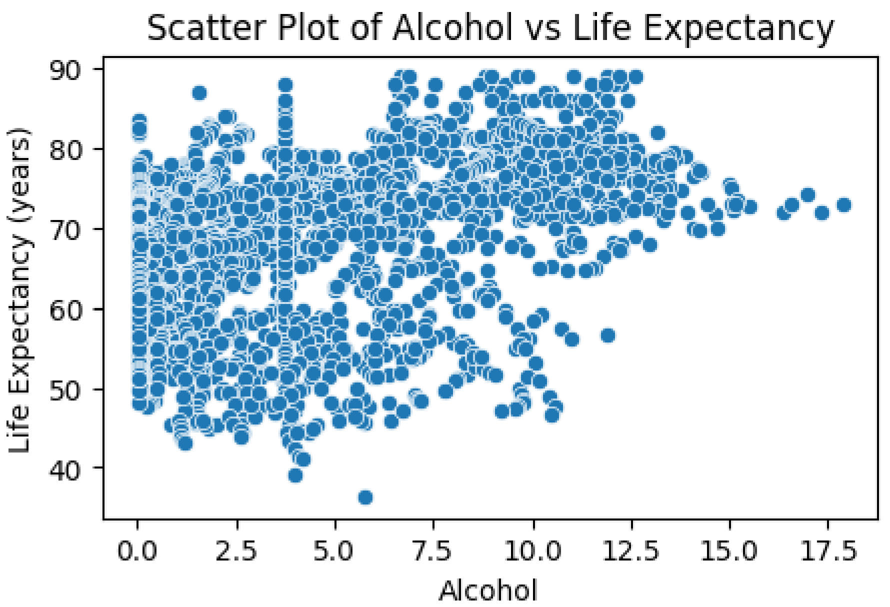


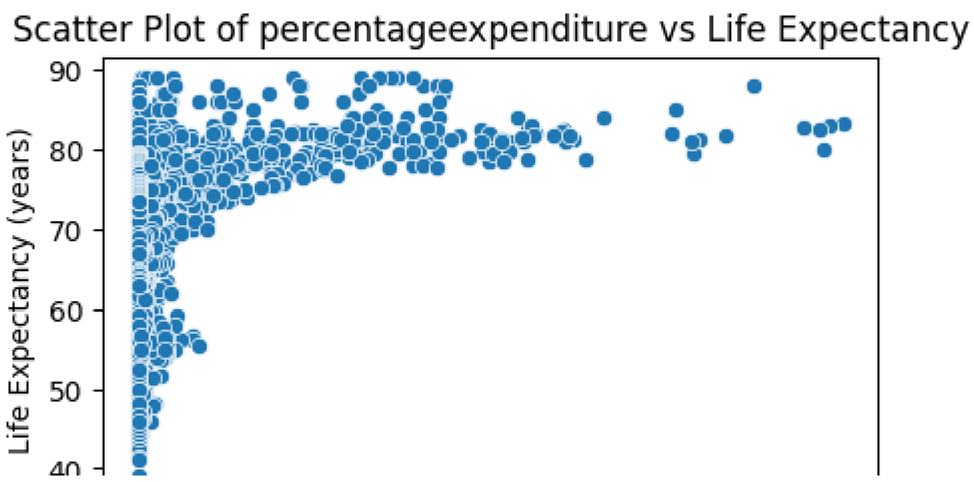

```
#plot scatter plots for each feature against Life Expectancy
for col in numerical_cols:
    if col != 'Lifeexpectancy':
        plt.figure(figsize=(5, 3))
        sns.scatterplot(x=df[col], y=df['Lifeexpectancy'])
        plt.title(f'Scatter Plot of {col} vs Life Expectancy')
        plt.xlabel(col)
        plt.ylabel('Life Expectancy (years)')
        plt.show()
```


Scatter Plot of Year vs Life Expectancy
Life Expectancy (years)
Year

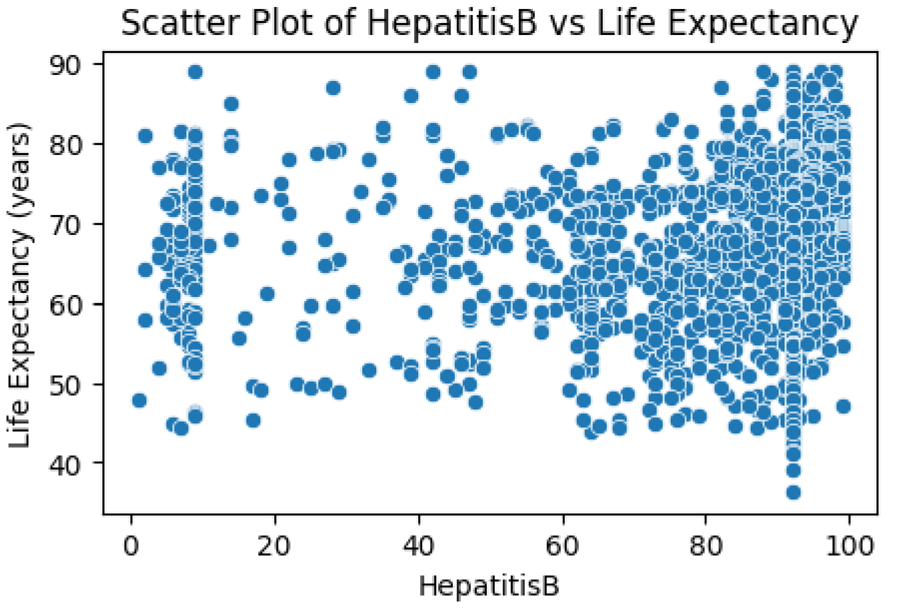

Scatter Plot of AdultMortality vs Life Expectancy
Life Expectancy (years)
AdultMortality

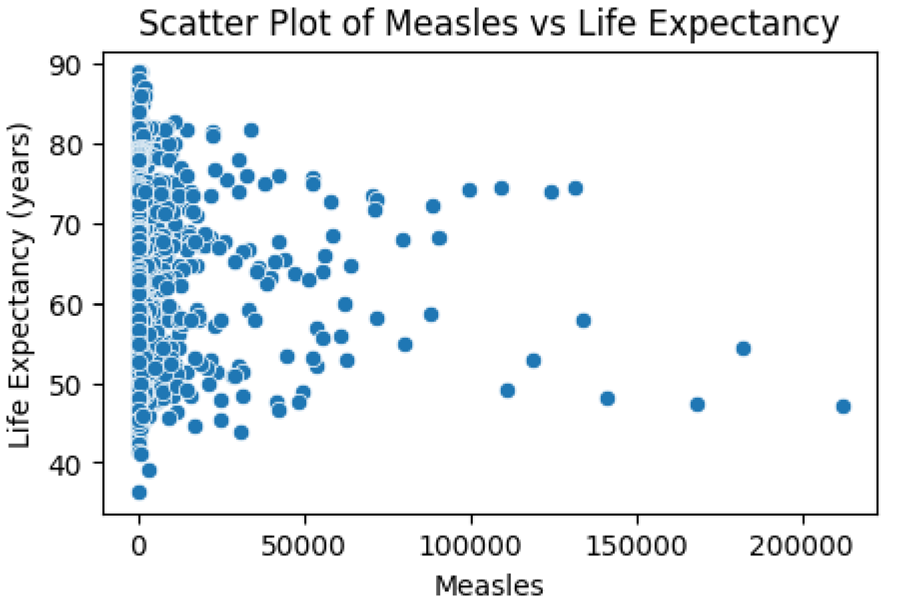

Scatter Plot of infantdeaths vs Life Expectancy
Life Expectancy (years)
infantdeaths

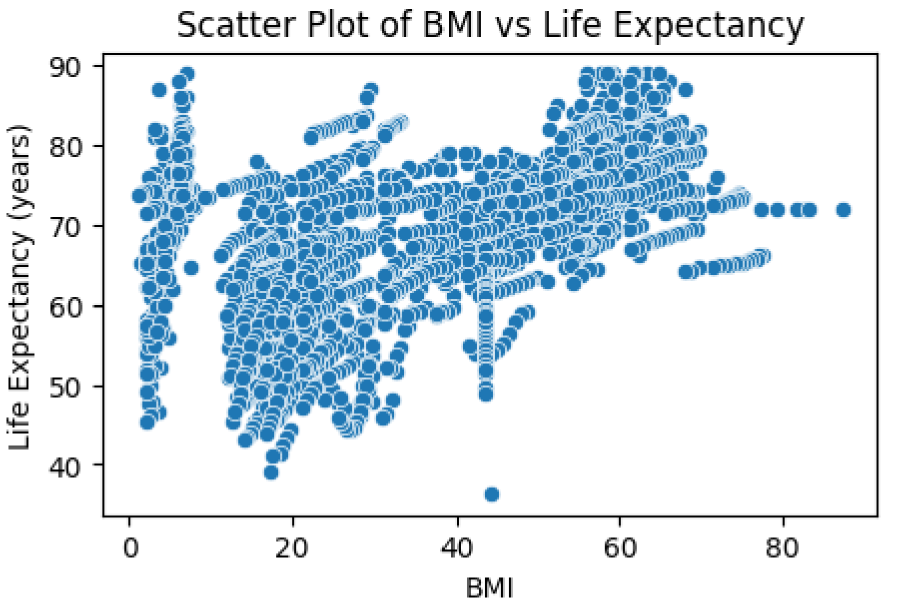

Scatter Plot of Alcohol vs Life Expectancy
Life Expectancy (years)
Alcohol

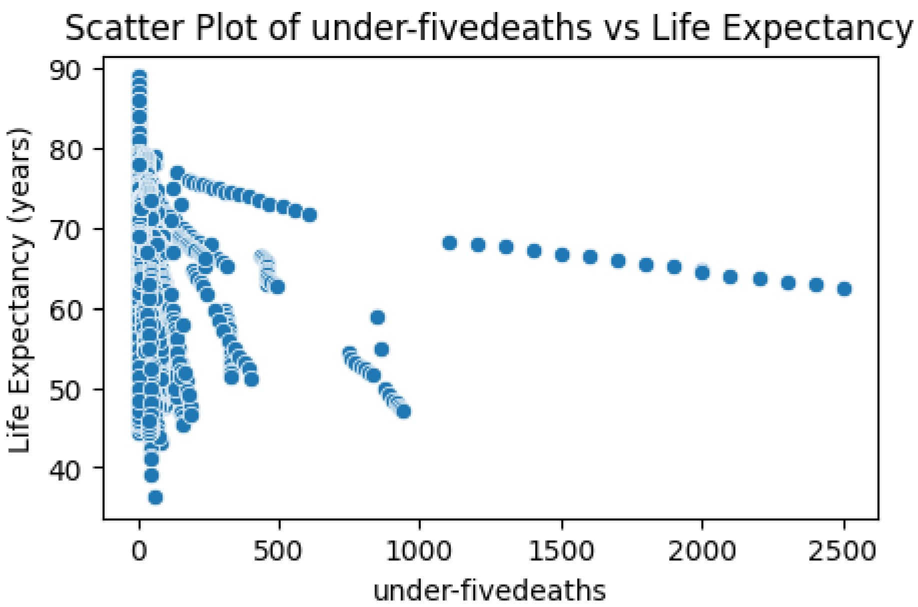

Scatter Plot of percentageexpenditure vs Life Expectancy
Life Expectancy (years)

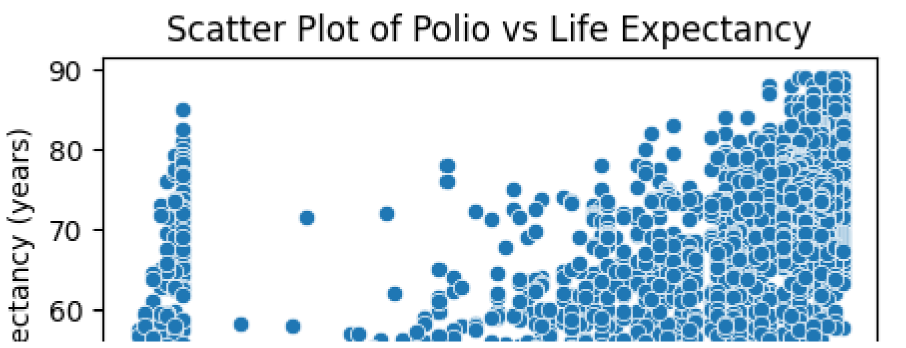

0 2500 5000 7500 10000 12500 15000 17500 20000
percentageexpenditure

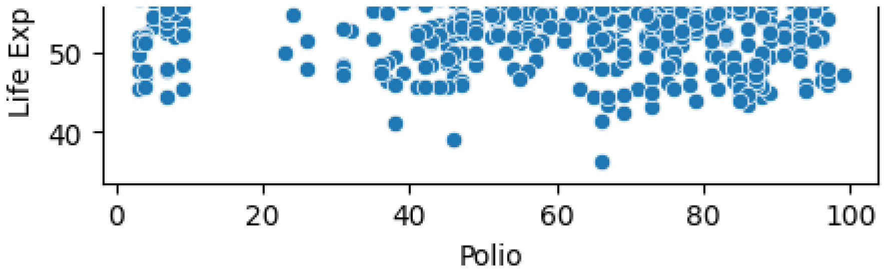

Scatter Plot of HepatitisB vs Life Expectancy
Life Expectancy (years)
HepatitisB


Scatter Plot of Measles vs Life Expectancy
Life Expectancy (years)
Measles

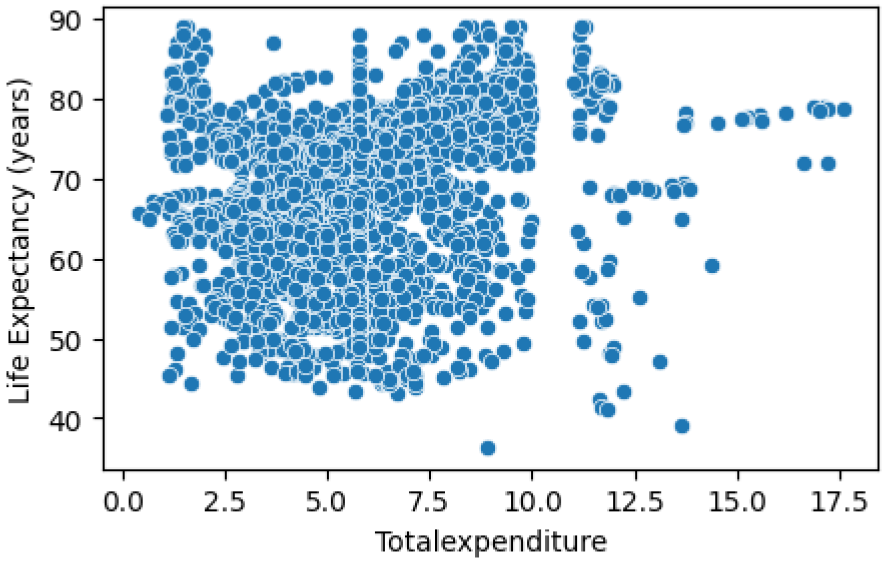

Scatter Plot of BMI vs Life Expectancy
Life Expectancy (years)
BMI


Scatter Plot of under-fivedeaths vs Life Expectancy
Life Expectancy (years)
under-fivedeaths

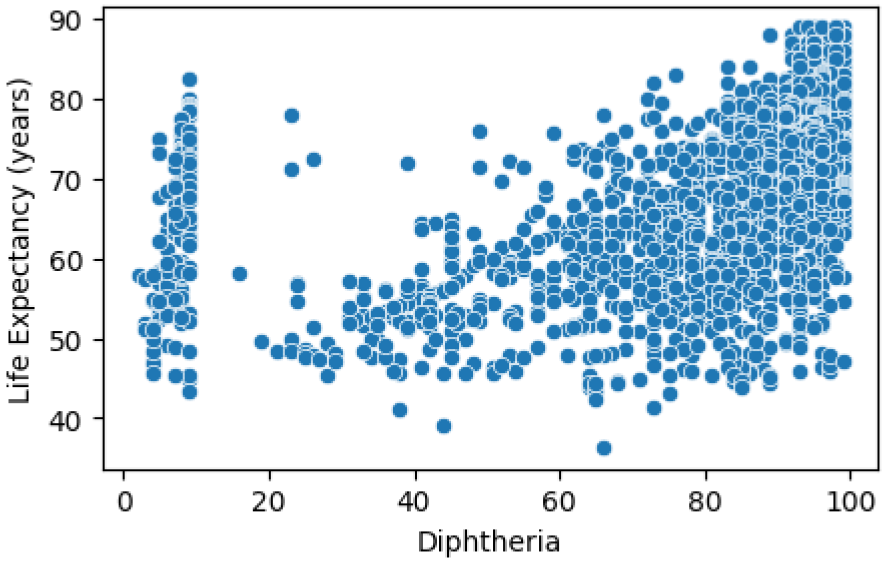

Scatter Plot of Polio vs Life Expectancy



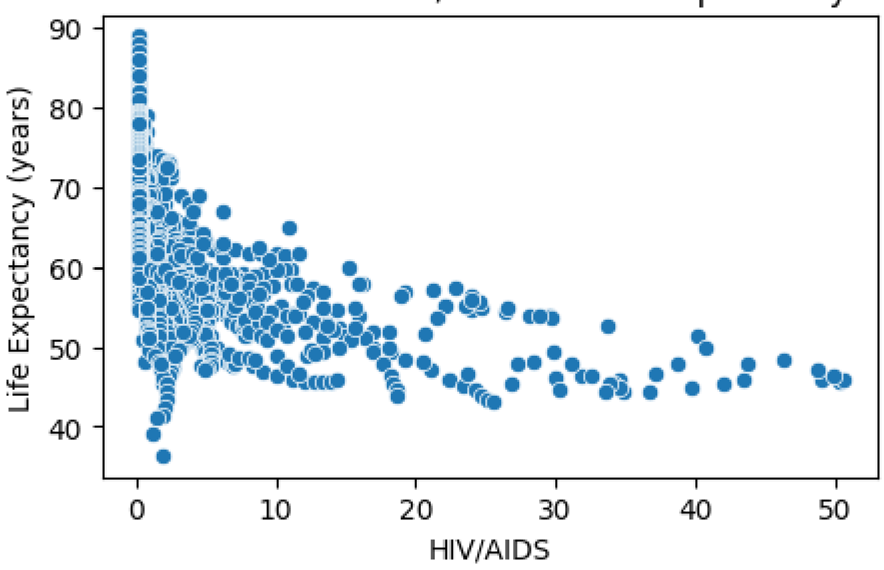
Scatter Plot of Totalexpenditure vs Life Expectancy



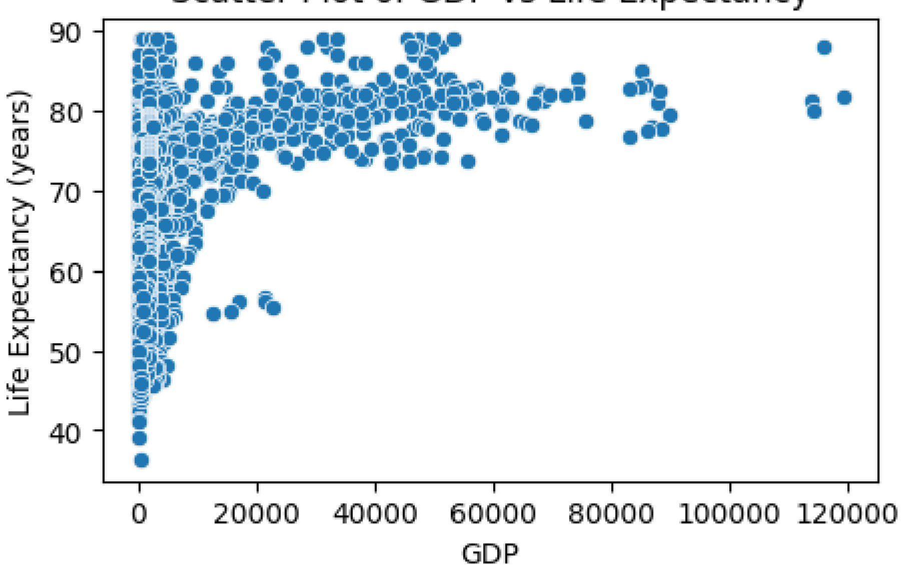
Scatter Plot of Diphtheria vs Life Expectancy



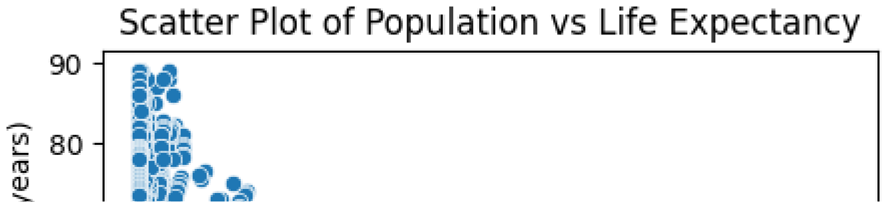
Scatter Plot of HIV/AIDS vs Life Expectancy

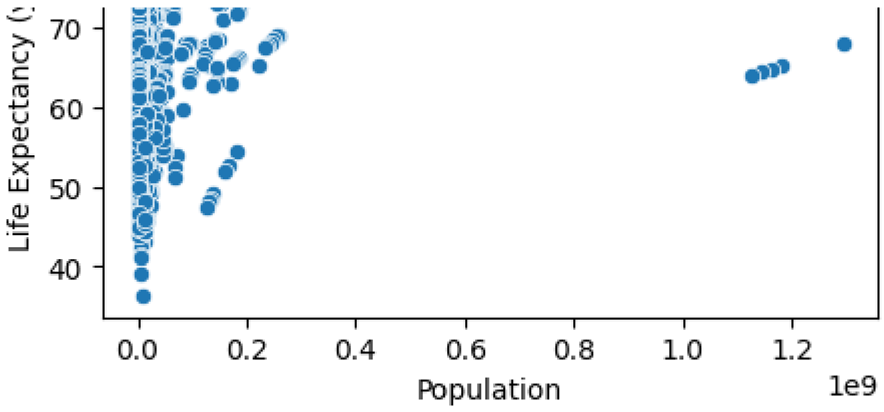


Scatter Plot of GDP vs Life Expectancy

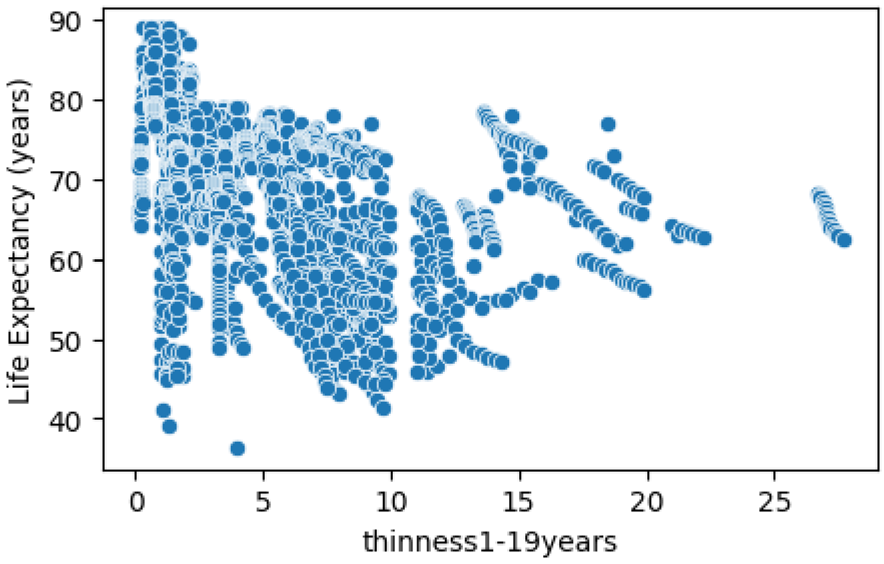


Scatter Plot of Population vs Life Expectancy

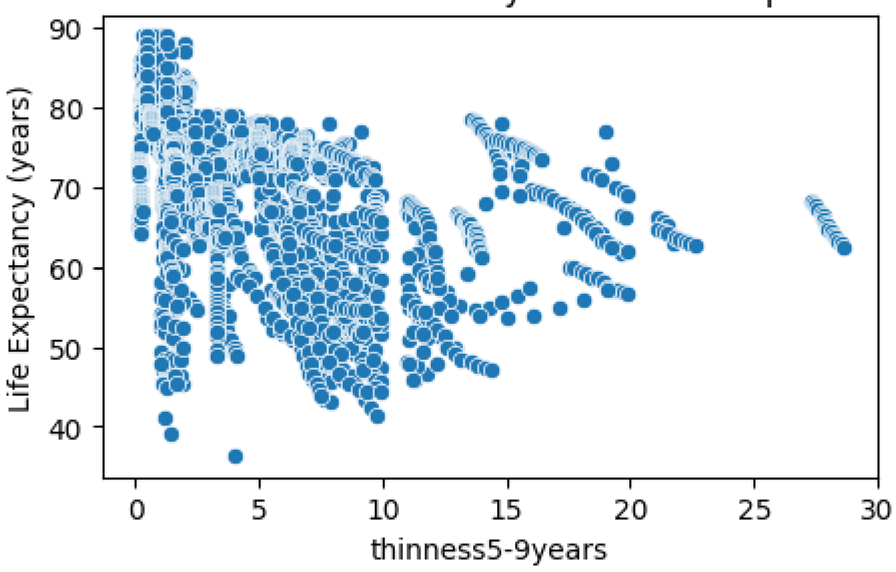




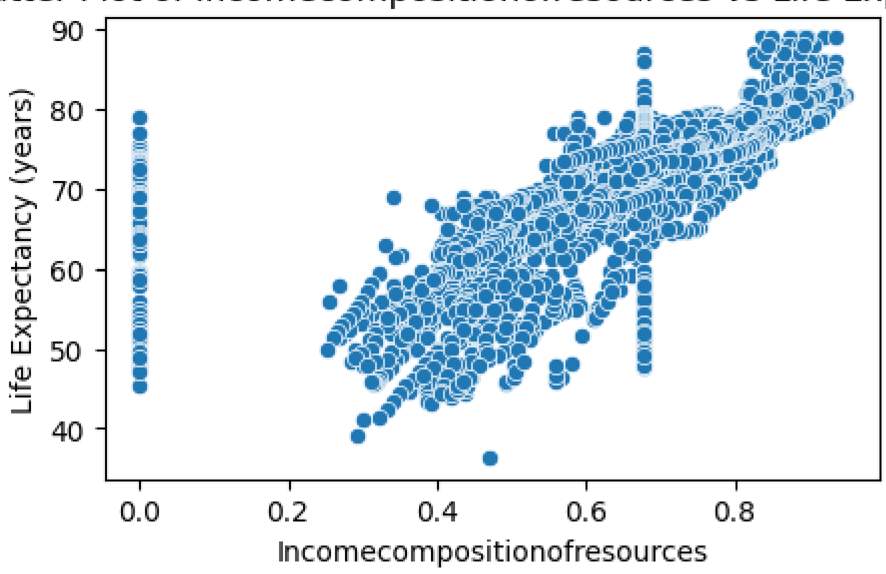
Scatter Plot of thinness1-19years vs Life Expectancy



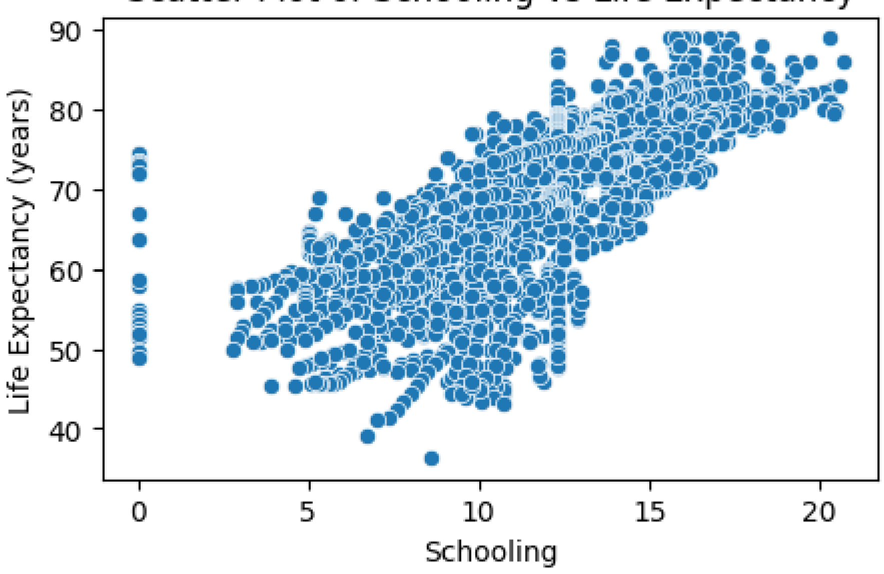
Scatter Plot of thinness5-9years vs Life Expectancy



Scatter Plot of Incomecompositionofresources vs Life Expectancy



Scatter Plot of Schooling vs Life Expectancy



Double-click (or enter) to edit

Feature Engineering

Encoding categorical variables (Country and Status)

```
df = pd.get_dummies(df, columns=['Status'], drop_first=True)  # Converts Status to binary (1 for Developed, 0 for Developing)
```

For Country, since there are many unique values, we'll drop it to avoid high dimensionality

```
df.drop('Country', axis=1, inplace=True)  # Removes Country column
```

Define features (X) and target (y)

```
X = df.drop('Lifeexpectancy', axis=1)  # All columns except Life expectancy
y = df['Lifeexpectancy']  # Target variable we want to predict
```

Split the data into training and testing sets (80% train, 20% test)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Feature Scaling

Many algorithms perform better when features are on similar scales

```
scaler = StandardScaler()  # Standardizes features by removing mean and scaling to unit variance
X_train_scaled = scaler.fit_transform(X_train)  # Fit to training data and transform
X_test_scaled = scaler.transform(X_test)  # Transform test data using same scaling
```

Model Building and Evaluation

Dictionary to store evaluation results

1. Linear Regression

```
results = {}

# 1. Linear Regression
lr = LinearRegression()  # Creates a linear regression model
lr.fit(X_train_scaled, y_train)  # Trains the model on scaled training data
y_pred_lr = lr.predict(X_test_scaled)  # Makes predictions on test data
```

Calculate evaluation metrics

```
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))  # Root Mean Squared Error
mae_lr = mean_absolute_error(y_test, y_pred_lr)  # Mean Absolute Error
r2_lr = r2_score(y_test, y_pred_lr)  # R-squared score
```

```
# Store results
results['Linear Regression'] = {'RMSE': rmse_lr, 'MAE': mae_lr, 'R2': r2_lr}
```

2. Random Forest Regressor

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)  # Creates RF model with 100 trees
rf.fit(X_train_scaled, y_train)  # Trains the model
y_pred_rf = rf.predict(X_test_scaled)  # Makes predictions
```

Double-click (or enter) to edit

Calculate metrics

```
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

Store results

```
results['Random Forest'] = {'RMSE': rmse_rf, 'MAE': mae_rf, 'R2': r2_rf}
```

3. XGBoost Regressor

```
xgb = XGBRegressor(n_estimators=100, random_state=42)  # Creates XGBoost model with 100 trees
xgb.fit(X_train_scaled, y_train)  # Trains the model
y_pred_xgb = xgb.predict(X_test_scaled)  # Makes predictions
```

Calculate metrics

```
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)
```

Store results

```
results['XGBoost'] = {'RMSE': rmse_xgb, 'MAE': mae_xgb, 'R2': r2_xgb}
```

4. Support Vector Regression

```
svr = SVR(kernel='rbf')  # Creates SVR model with Radial Basis Function kernel
svr.fit(X_train_scaled, y_train)  # Trains the model
y_pred_svr = svr.predict(X_test_scaled)  # Makes predictions
```

Calculate metrics

```
rmse_svr = np.sqrt(mean_squared_error(y_test, y_pred_svr))
mae_svr = mean_absolute_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)
```

Store results

```
results['Support Vector Regression'] = {'RMSE': rmse_svr, 'MAE': mae_svr, 'R2': r2_svr}
```

Display results

```
print("\nModel Evaluation Results:")
for model, metrics in results.items():
    print(f"\n{model}:")
    print(f"RMSE: {metrics['RMSE']:.2f}")  # Lower is better
    print(f"MAE: {metrics['MAE']:.2f}")  # Lower is better
    print(f"R2 Score: {metrics['R2']:.2f}")  # Closer to 1 is better
```

```
Model Evaluation Results:

Linear Regression:
RMSE: 3.91
MAE: 2.86
R2 Score: 0.82

Random Forest:
RMSE: 1.66
MAE: 1.08
R2 Score: 0.97

XGBoost:
RMSE: 1.75
MAE: 1.18
R2 Score: 0.96

Support Vector Regression:
RMSE: 3.32
MAE: 2.31
R2 Score: 0.87
```

Feature Importance for the best performing model (Random Forest or XGBoost)

Let's check feature importance from Random Forest

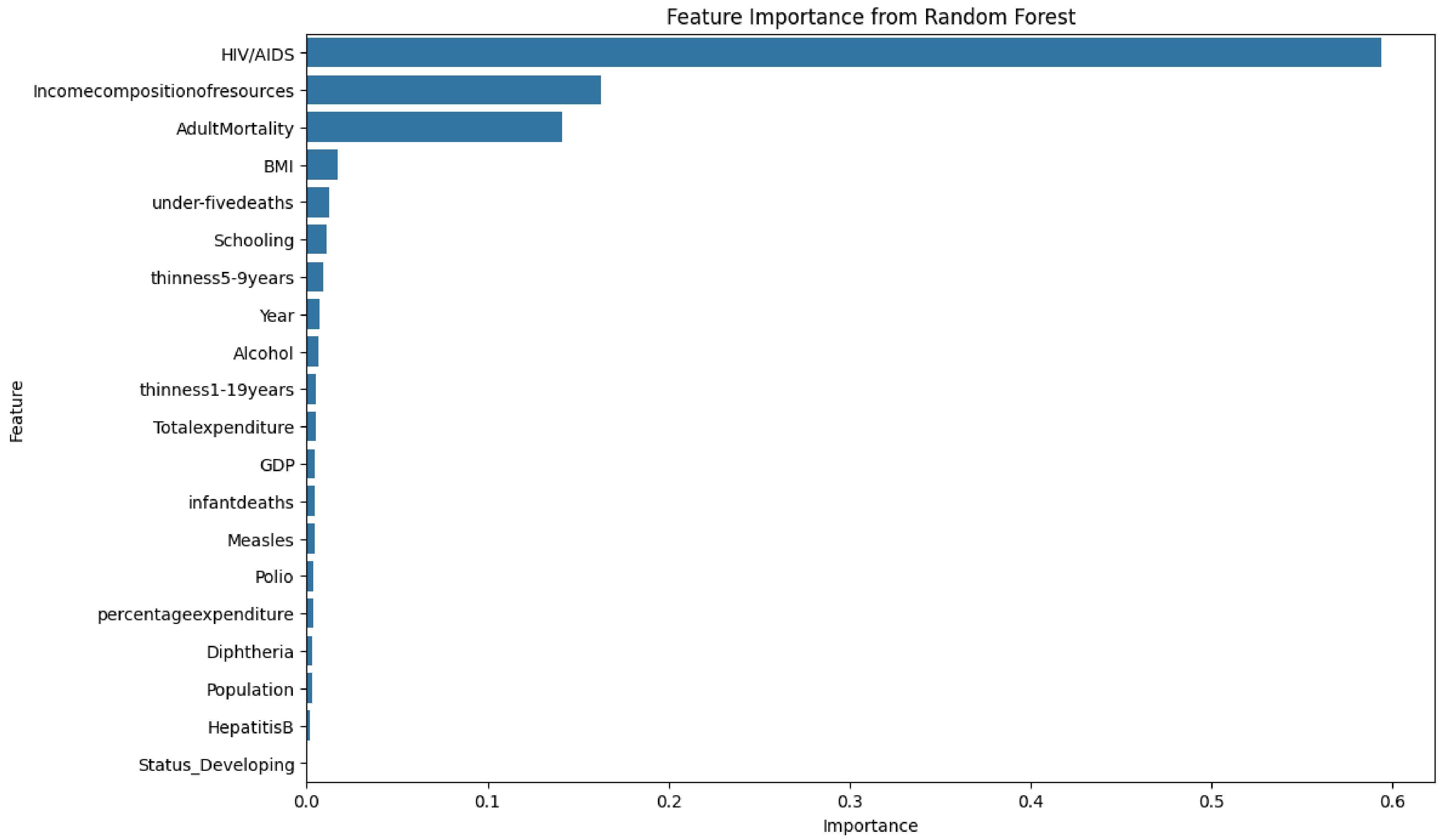
```
feature_importance = rf.feature_importances_  # Gets importance scores from trained RF model
features = X.columns  # Gets feature names
```

Create a dataframe for visualization

```
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importance})
importance_df = importance_df.sort_values('Importance', ascending=False)  # Sorts by importance
```

Plot feature importance

```
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance from Random Forest')
plt.show()
```



gdp per capita

```
#implement simple linear regression to predict life expectancy based solely on GDP per capita
df['gdp_per_capita'] = df['GDP'] / df['Population']  # Calculate GDP per capita
#MULTIPLY GDP per capita by 1000 to scale it
df['gdp_per_capita'] *= 1000000  # Scale GDP per capita for better interpretability
print("\nGDP per Capita:")
print(df['gdp_per_capita'])  # Display first few rows of GDP per capita
```

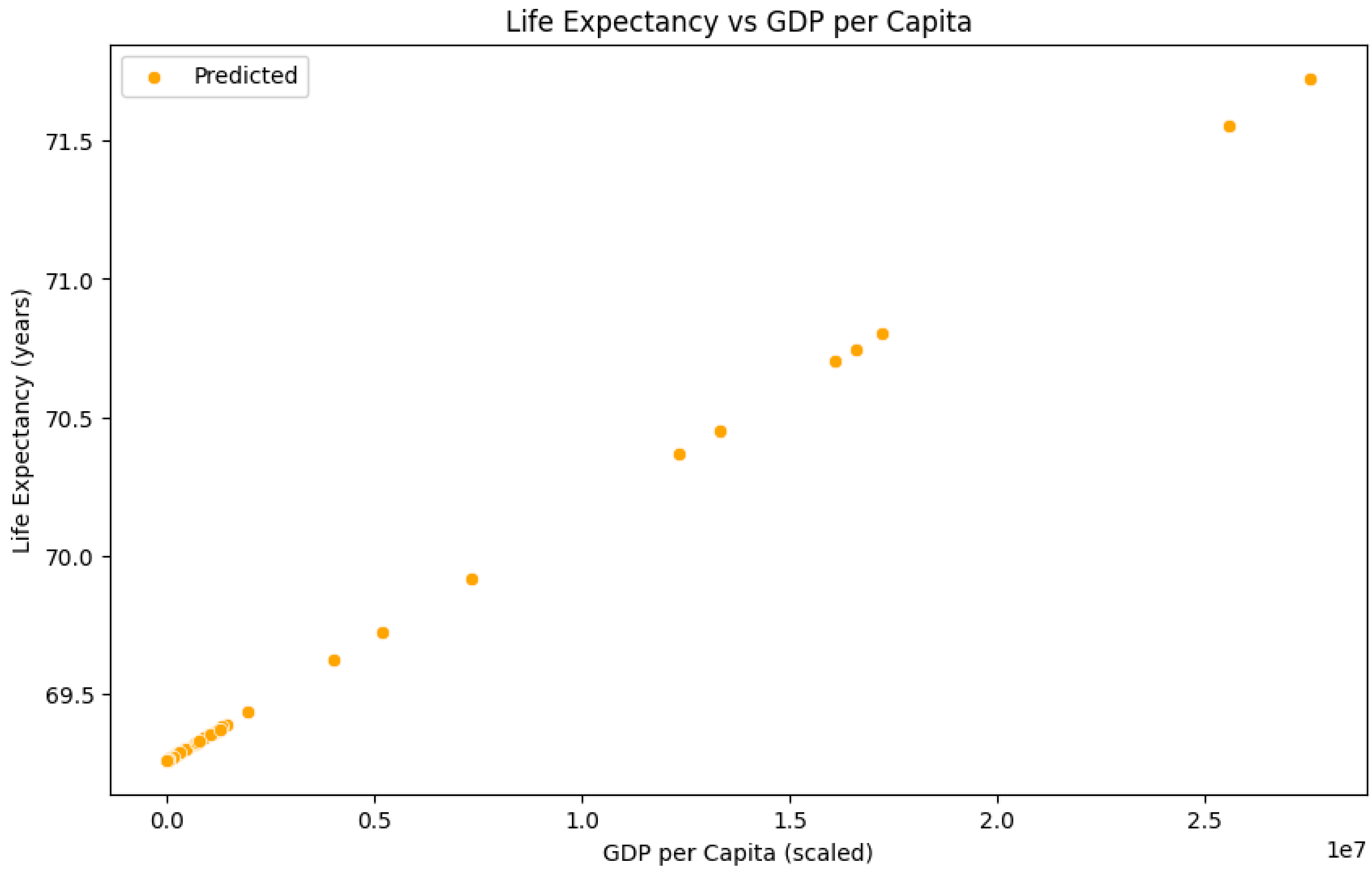
```
GDP per Capita:
0         17.318314
1       1870.360746
2         19.908962
3        181.218991
4         21.331247
            ...
2933      35.559872
2934      35.883715
2935     456.867875
2936      44.361960
2937      44.783803
Name: gdp_per_capita, Length: 2938, dtype: float64
```

```
x = df[['gdp_per_capita']]  # Feature: GDP per capita
y = df['Lifeexpectancy']  # Target: Life expectancy
from sklearn.model_selection import train_test_split  # Import for splitting data
from sklearn.linear_model import LinearRegression  # Import for linear regression
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
rmse_simple_lr = np.sqrt(mean_squared_error(y_test, y_pred))
mae_simple_lr = mean_absolute_error(y_test, y_pred)
print("\nSimple Linear Regression Results:")
print(f"RMSE: {rmse_simple_lr:.2f}")
print(f"MAE: {mae_simple_lr:.2f}")

plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(x=x_test['gdp_per_capita'], y=y_pred, label='Predicted', color='orange')
plt.title('Life Expectancy vs GDP per Capita')
plt.xlabel('GDP per Capita (scaled)')
plt.ylabel('Life Expectancy (years)')
plt.legend()
```

```
Simple Linear Regression Results:
RMSE: 9.29
MAE: 7.63
<matplotlib.legend.Legend at 0x256c2223910>
```




```
# random forest regression to predict life expectancy based on GDP per capita
rf_simple = RandomForestRegressor(n_estimators=100, random_state=42)
rf_simple.fit(x_train, y_train)  # Train the model
y_pred_rf_simple = rf_simple.predict(x_test)  # Make predictions
rmse_rf_simple = np.sqrt(mean_squared_error(y_test, y_pred_rf_simple))  # Calculate RMSE
mae_rf_simple = mean_absolute_error(y_test, y_pred_rf_simple)  # Calculate MAE
print("\nRandom Forest Regression Results:")
print(f"RMSE: {rmse_rf_simple:.2f}")  # Print RMSE
print(f"MAE: {mae_rf_simple:.2f}")  # Print MAE
plt.figure(figsize=(10, 6))
sns.scatterplot(x=x_test['gdp_per_capita'], y=y_pred_rf_simple, label='Predicted', color='green')
plt.title('Life Expectancy vs GDP per Capita (Random Forest)')
plt.xlabel('GDP per Capita (scaled)')
plt.ylabel('Life Expectancy (years)')
plt.legend()
plt.show()  # Show the plot
```

```
Random Forest Regression Results:
RMSE: 9.77
MAE: 7.71
```