Assignment 1

Introduction to ML

COSC2673 Machine Learning

(Undergraduate Level)

By Oisin Sol Emlyn Aeonn

Student ID: s3952320

1.0 Table of Contents

7.4 Ensemble Voting

1.1 Read Me

- All tasks were completed in accordance with the Assignment 1 Brief.
- The Discussion Forum, Tutorials, and Lectures were all utilized effectively to get as best as an understanding of the assignment requirements as possible, however there was some ambiguity, and freedom in the way things were phrased, so I tried to have as much fun, and treat this as a real-world problem that would have implications if I was to get things wrong.
- A Complementary Section has been added to cover more advanced topics for fun, and for my own personal learning.
- All predictions, including those in `predictions.csv`, are generated dynamically within this Jupyter Notebook.
- This project was completed in alignment with RMIT University's Academic Integrity Policy and was solely undertaken by Oisin Sol Emlyn Aeonn (s3952320) within the specified timeframe. All materials used are cited, and referenced accordingly.
- The project was submitted on-time to the COSC2673 (Undergraduate) Machine Learning Canvas before 5PM Monday, 8th of April 2024.
- A Video can be found here: X going through my presentation of this Jupyter Notebook, and corresponding materials.
- I am deeply passionate about Machine Learning and aspire to build a career in this field. I am eager for any constructive feedback to enhance my learning and application.
- Special thanks to Dr Azadeh Alavi, Dr Pubudu Sanjeevani, and Ms. Rumin Chu for their invaluable assistance and support in my learning journey.
- I hope you enjoy my work!

1.2 Introduction

- **Problem Statement:**
  - The goal is to develop a predictive model for Life Expectancy using a modified WHO dataset, focusing on understanding key factors that influence it.
  - Starting with a baseline model incorporating all features, subsequent optimization will be pursued through advanced machine learning techniques.
- **Dataset:**
  - The analysis is based on a modified version of the WHO Life Expectancy dataset from Kaggle, encompassing various indicators that could impact life expectancy across nations.
- **Objectives:**
  - Master the core principles of Machine Learning applicable to this context.
  - Perform comprehensive Exploratory Data Analysis (EDA) to uncover insights within the dataset.
  - Implement data preprocessing to ensure the dataset's integrity for modeling.
  - Iteratively refine the predictive model to enhance its accuracy in forecasting Life Expectancy.
  - Apply and interpret evaluation metrics to systematically improve the model.
- **Scope:**
  - Focus on predicting Life Expectancy leveraging methods and insights from weeks 1-4 of the COSC2673 Machine Learning Course.

- Adopt strategies such as regularization and normalization to refine the model, maintaining the integrity of the feature set.
- Strive for the optimal model performance within the established constraints.
- Undertake predictions on a separate, unseen dataset (`predictions.csv`) to validate the model's generalisability.

1.3 Python Library Imports

- First, let's cover all of the required imports for this entire Jupyter Notebook.
- I put them all here so that you can run most cells (especially EDA) out of order.
- This also conforms to the coding standard DO NOT REPEAT YOURSELF (DRY).
- I also have provided a commented description of each library for their use.

In [2]:
```
#
import os

#
import warnings
warnings.filterwarnings('ignore')

#
import numpy as np

# This package
import pandas as pd

#
import seaborn as sns

# Used for some things that
import matplotlib.pyplot as plt

# Native support, and offers interactive graphs
import plotly.express as px

# Debug
print(os.environ['PATH'])

plt.style.use('dark_background')

import plotly.express as px
```

```
/opt/anaconda/bin:/opt/anaconda/condabin:/usr/local/sbin:/usr/local/bin:/usr/bin:/var/lib/flatpak/exports/bin:/usr/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl:/var/lib/snapd/snap/bin
```

2.0 Data Ingestion

- In this step, I will ingest the data into a Pandas DataFrame named lifeExpectancyFrame.
- DataFrames allow us to efficiently explore and manipulate data.
- It is important to note that it is bad practice to modify our original DataFrame.

In [3]:
```
# File format doesn't require any delimiters :)
lifeExpectancyFrame = pd.read_csv('./dataset/train.csv')

# Check data is loaded properly into the Pandas DataFrame
lifeExpectancyFrame.head(3)
```

Out[3]:

	ID	TARGET_LifeExpectancy	Country	Year	Status	AdultMortality	AdultMortality-Male	AdultMortality-Female	SLS	Alcohol
0	1	67.1	146	2017	0	263	262	264	62	0.01
1	2	59.8	146	2016	0	271	278	264	64	0.01
2	3	57.6	146	2015	0	268	246	290	66	0.01

3 rows × 24 columns

2.1 Exploratory Data Analysis (EDA)

In this step, I will examine the data in the Pandas DataFrame.

Let's start with getting how many rows, and columns are in our dataset!

In [4]:
```
# Size of the data (rows, columns)
lifeExpectancyFrame.shape
```

Out[4]: (2071, 24)

As you can see we have 2071 rows (instances) of data that we can play around with, now let's check what type of values we have!

In [5]:
```
# Information about the Data like Data Type, null values, etc.
lifeExpectancyFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ID                              2071 non-null   int64
 1   TARGET_LifeExpectancy           2071 non-null   float64
 2   Country                         2071 non-null   int64
 3   Year                            2071 non-null   int64
 4   Status                          2071 non-null   int64
 5   AdultMortality                  2071 non-null   int64
 6   AdultMortality-Male             2071 non-null   int64
 7   AdultMortality-Female           2071 non-null   int64
 8   SLS                             2071 non-null   int64
 9   Alcohol                         2071 non-null   float64
 10  PercentageExpenditure           2071 non-null   float64
 11  Measles                         2071 non-null   int64
 12  BMI                             2071 non-null   float64
 13  Under5LS                        2071 non-null   int64
 14  Polio                           2071 non-null   int64
 15  TotalExpenditure                2071 non-null   float64
 16  Diphtheria                      2071 non-null   float64
 17  HIV-AIDS                        2071 non-null   float64
 18  GDP                             2071 non-null   float64
 19  Population                      2071 non-null   int64
 20  Thinness1-19years               2071 non-null   float64
 21  Thinness5-9years                2071 non-null   float64
 22  IncomeCompositionOfResources    2071 non-null   float64
 23  Schooling                       2071 non-null   float64
dtypes: float64(12), int64(12)
memory usage: 388.4 KB
```

- As seen above, our data is free from null values.
- However, the absence of null values isn't indicative that there aren't any incorrect values.
- We will identify any incorrect values soon.
- But let's now check if we have any duplicate rows in our data.

In [6]:
```
# Check if there are any duplicated rows in the DataFrame
has_duplicates = lifeExpectancyFrame.duplicated().any()

# Print out the result
print(f"DuplicateData: {has_duplicates}")
```

```
DuplicateData: False
```

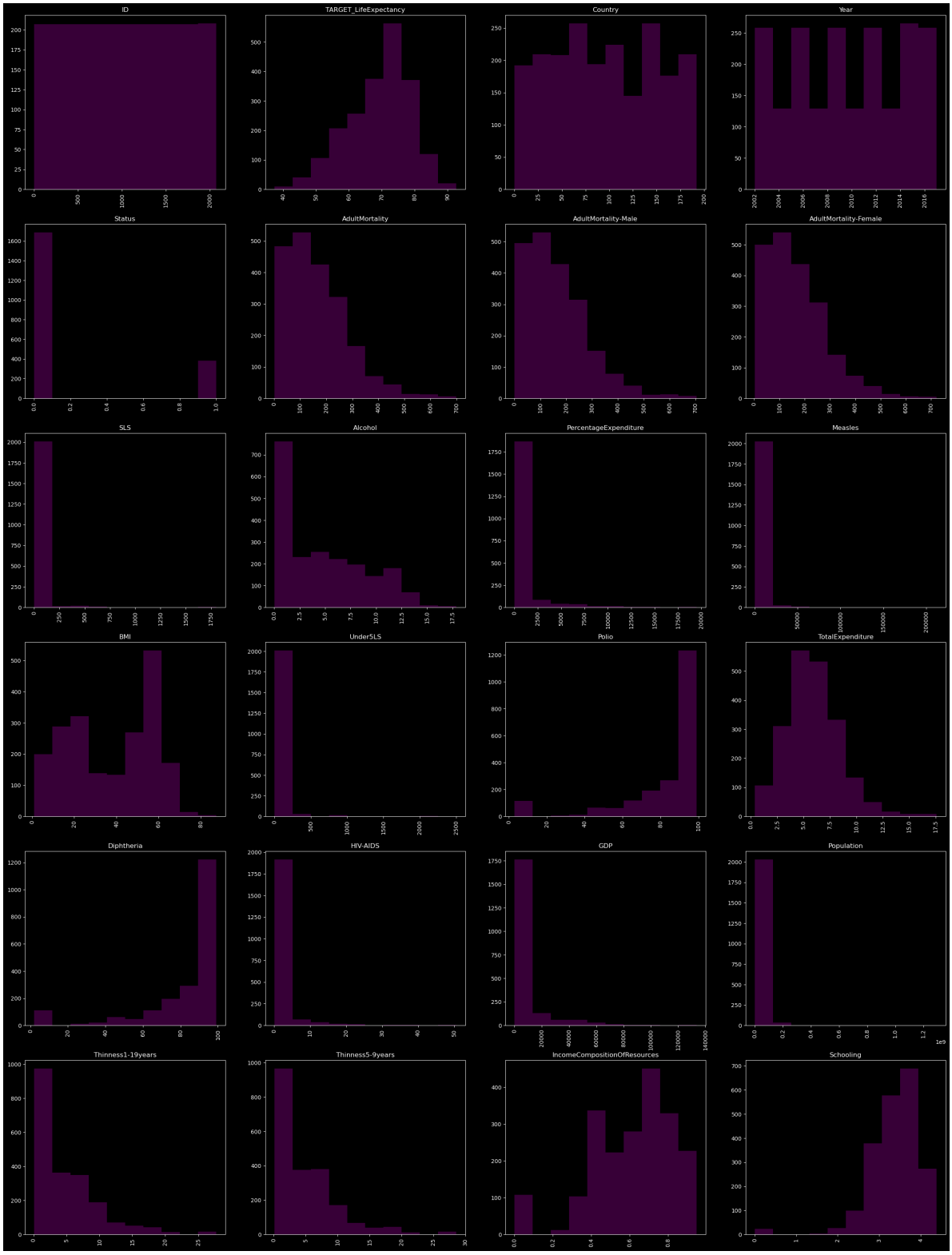
In [7]:
```
# Temporarily adjust the Pandas display options within the context manager (Data has too many columns to fit)
with pd.option_context('display.max_columns', None):  # None means unlimited
    display(lifeExpectancyFrame.describe())
```

	ID	TARGET_LifeExpectancy	Country	Year	Status	AdultMortality	AdultMortality-Male	A
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	
mean	1036.000000	69.274505	95.360212	2009.518590	0.185418	162.833897	161.908257	
std	597.990524	9.482281	54.861641	4.614147	0.388730	118.872170	119.442235	
min	1.000000	37.300000	0.000000	2002.000000	0.000000	1.000000	0.000000	
25%	518.500000	63.000000	50.000000	2006.000000	0.000000	74.000000	74.000000	
50%	1036.000000	71.200000	94.000000	2010.000000	0.000000	144.000000	142.000000	
75%	1553.500000	76.000000	144.000000	2014.000000	0.000000	228.000000	228.000000	
max	2071.000000	92.700000	192.000000	2017.000000	1.000000	699.000000	704.000000	

- Here we can see all of the columns of our data.
- We are also presented with very important metrics for each column, such as median, max, min, and IQR (25th and 75th percentiles), among others.
- This information will be critical for removing outliers.
- It also aids in verifying the effectiveness of our data transformations.

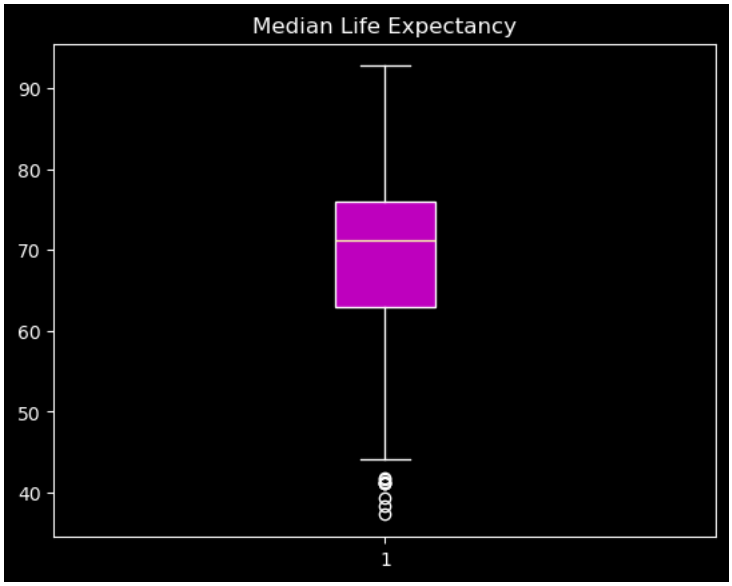
- **Identified Outliers:** Outliers were detected in the majority of dataset columns by comparing the IQR 25% and 75% to the maximum values in the following:
  - AdultMortality
  - AdultMortality-Female
  - AdultMortality-Male
  - SLS
  - Alcohol
  - PercentageExpenditure
  - Measles (very large outliers)
  - BMI
  - Under5LS
  - TotalExpenditure
  - HIV-AIDS (very large outliers)
  - GDP (large outliers)
  - Population (large outliers)
  - Thinness1-19years
  - Thinness5-9years

In [8]:
```
plt.figure(figsize=(30,40))
for i, col in enumerate(lifeExpectancyFrame.columns):
    plt.subplot(6,4,i+1)
    plt.hist(lifeExpectancyFrame[col], alpha=0.3, color='m')
    plt.title(col)
    plt.xticks(rotation='vertical')
```

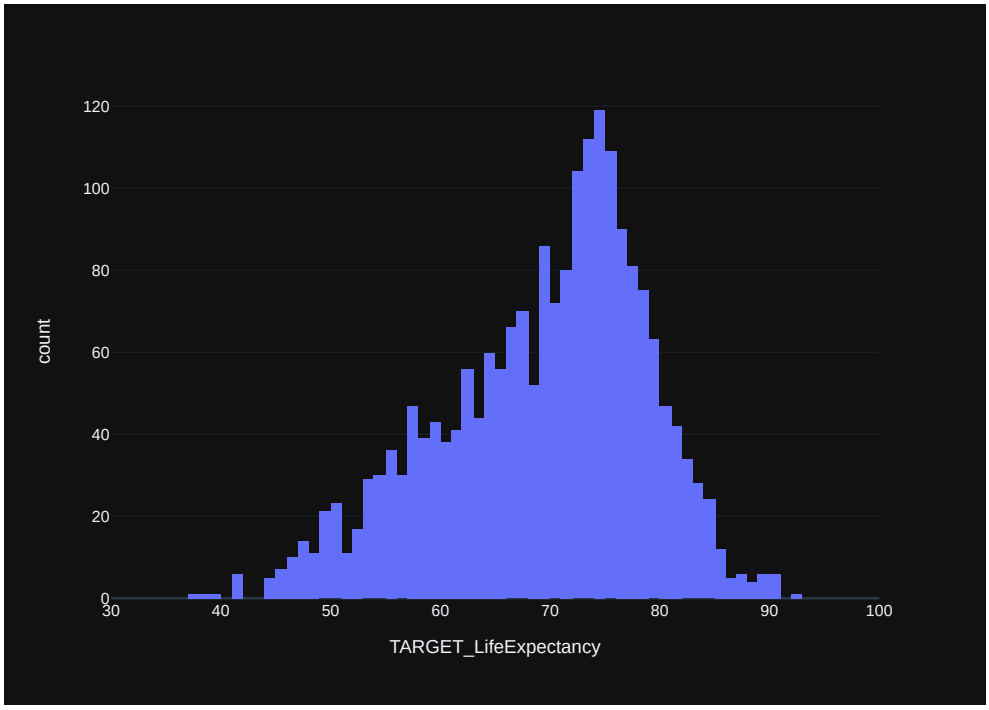


- These histograms show the distributions of all our columns, allowing us to observe if they are skewed, gaussian, multi-modal, etc.
- We can identify obvious outliers or disparities in the data.
- They enable a closer comparison of the value representations in the data, such as how large or small the values are.
- Given the considerable variation in our data, I hypothesis that transformations will definitely lead to improvements in our prediction metrics.

In [9]:
```
# Creating a boxplot
plt.boxplot(lifeExpectancyFrame['TARGET_LifeExpectancy'], patch_artist=True, boxprops=dict(facecolor='m', color=

# Adding title and customizing the plot
plt.title('Median Life Expectancy')

plt.show()
```

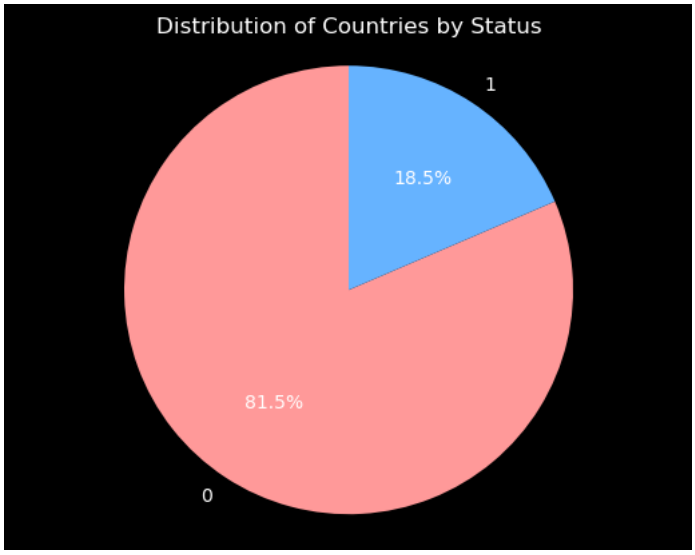


In [10…
```
# Create the histogram
fig = px.histogram(lifeExpectancyFrame, x='TARGET_LifeExpectancy', template='plotly_dark')

fig.update_xaxes(range=[30, 100])

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

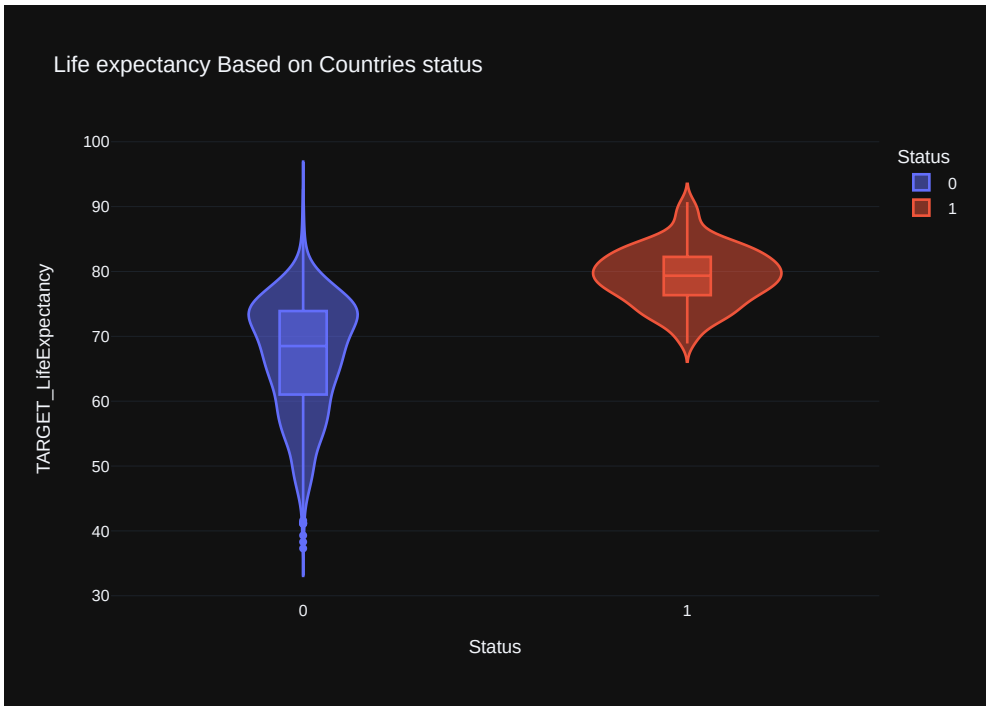


- Here I have created a boxplot and an interactive histogram of our target value, Life Expectancy.
- As observed, there are some outliers denoted in the visualizations.
- The distribution of Life Expectancy is near normal (Gaussian).

In [11…
```
# Count the occurrences of each status
status_counts = lifeExpectancyFrame['Status'].value_counts()

# Create a pie chart
fig, ax = plt.subplots()
ax.pie(status_counts, labels=status_counts.index, autopct='%1.1f%%', startangle=90, colors=['#ff9999','#66b3ff']

# Equal aspect ratio ensures that pie is drawn as a circle.
ax.axis('equal')

plt.title('Distribution of Countries by Status')
plt.show()
```

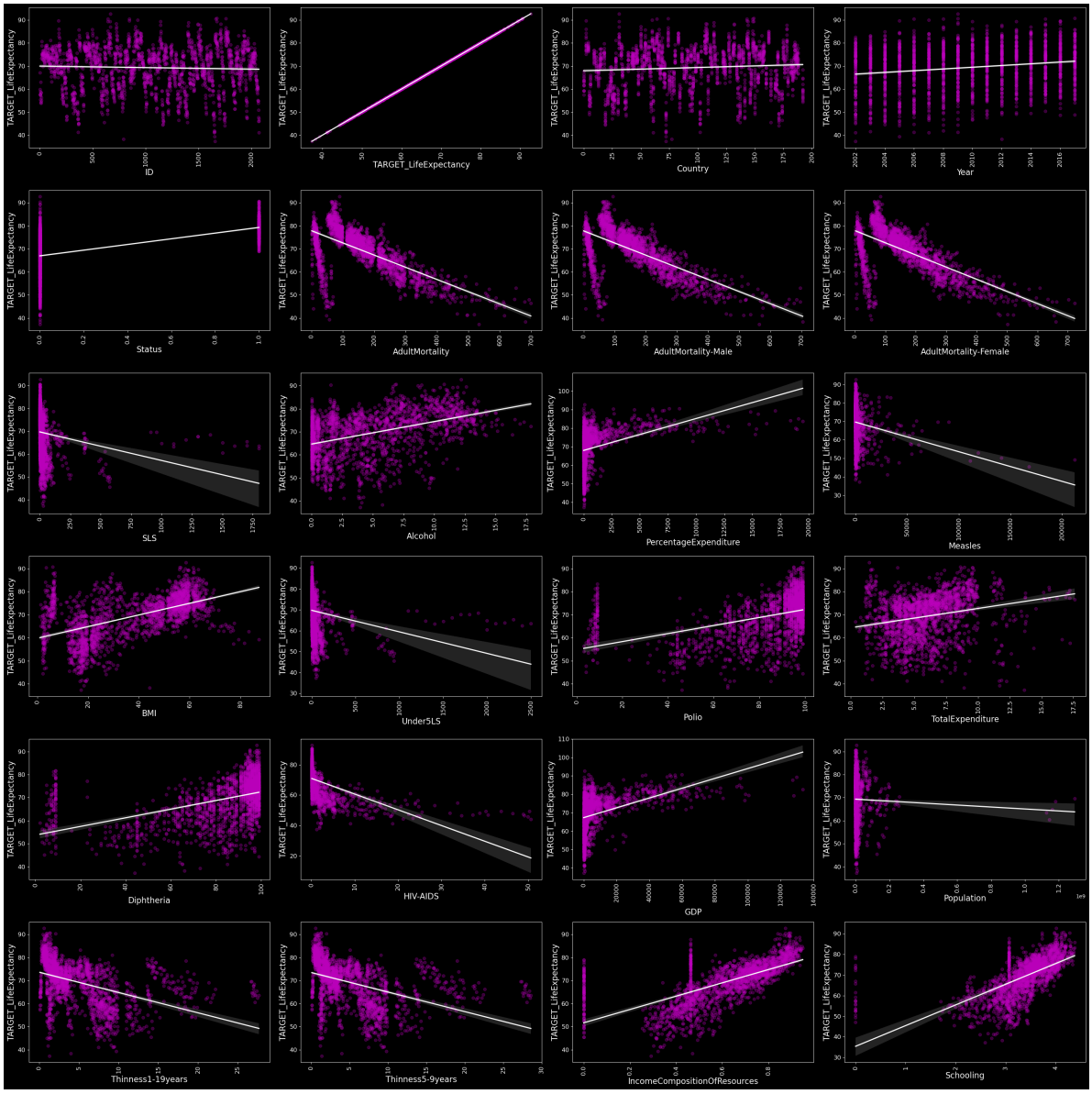


In [12…
```
fig=px.violin(lifeExpectancyFrame,x='Status',y='TARGET_LifeExpectancy',color='Status',template='plotly_dark',box
fig.show()
```



- I created a simple pie, and interactive violin plot to more closely examine the 'Status' attribute.
- It is clear from the chart that a larger number of countries in the world are considered to be Developing rather than Developed.
- It would be interesting to create a Logistic Model to predict whether a country is Developed or not based on Life Expectancy.
- Such a model could help us identify the decision boundary for classifying countries' development status based on Life Expectancy.

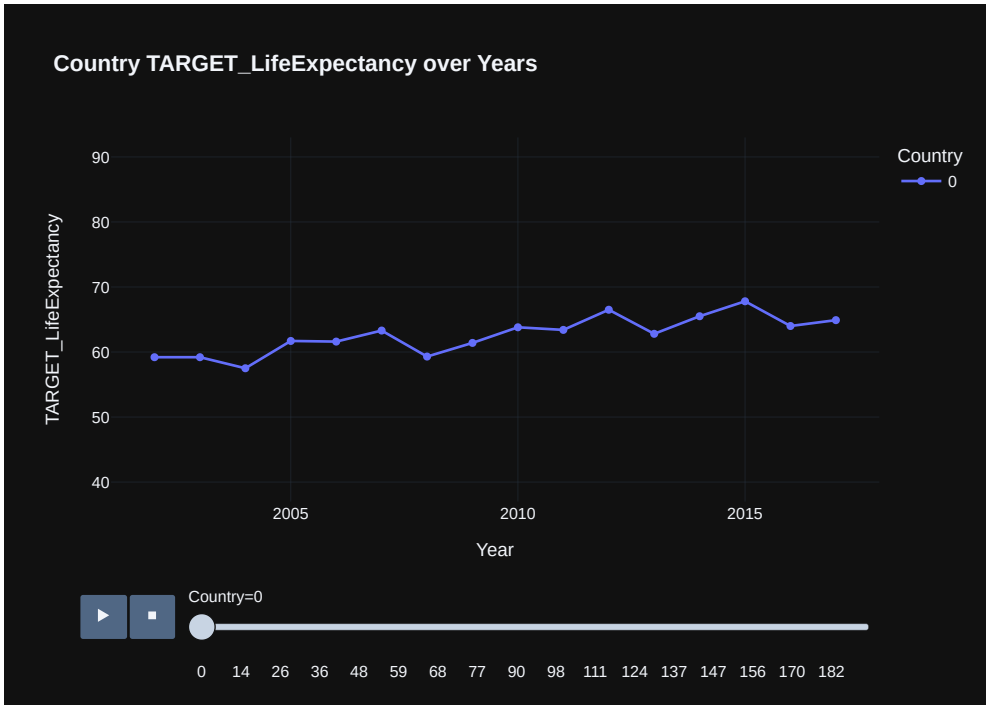
In [13…
```
plt.figure(figsize=(30, 30))

for i, col in enumerate(lifeExpectancyFrame.columns):
    plt.subplot(6, 4, i + 1)

    # Using regplot which combines scatterplot with regression line
    sns.regplot(x=col, y='TARGET_LifeExpectancy', data=lifeExpectancyFrame, scatter_kws={'alpha':0.3, 'color':'r

    plt.xticks(rotation='vertical', fontsize=12)
    plt.yticks(fontsize=12)
    plt.xlabel(col, fontsize=16)
    plt.ylabel('TARGET_LifeExpectancy', fontsize=16)

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



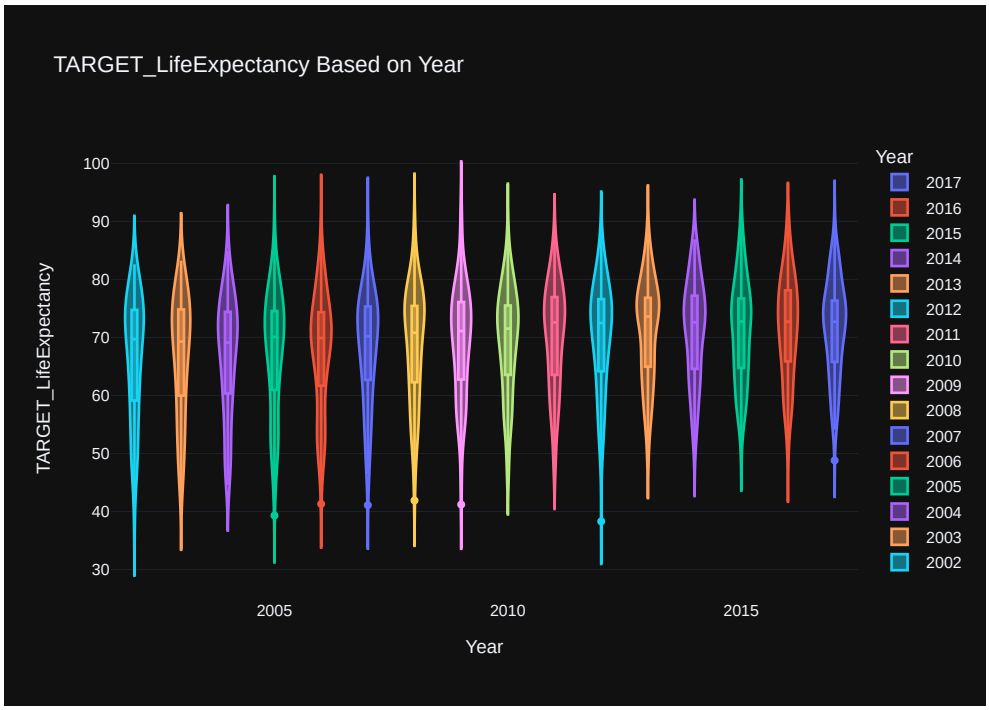
- Above is a key figure we must consider, using regplot (which combines regression/trend analysis with a scatterplot) to graph trends between two variables.
- In our case, I've used this to plot trends between our target, Life Expectancy, and all other columns.
- As we can observe, there are quite a few trends, which is promising as it suggests we should be able to achieve relatively high prediction scores.
- It is important to note which indicators are not good predictors (i.e., not correlated to the target), such as ID and Country.

In [14…

```
# Sort the DataFrame first by 'Country' and then by 'Year' within each country
sorted_lifeExpectancyFrame = lifeExpectancyFrame.sort_values(by=['Country', 'Year'])

# Create the line plot
fig = px.line(sorted_lifeExpectancyFrame, x='Year', y='TARGET_LifeExpectancy',
              animation_frame='Country', animation_group='Year',
              color='Country', markers=True, template='plotly_dark',
              title='<b>Country TARGET_LifeExpectancy over Years</b>')

# Manually set the y-axis range
fig.update_yaxes(range=[37, 93])

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

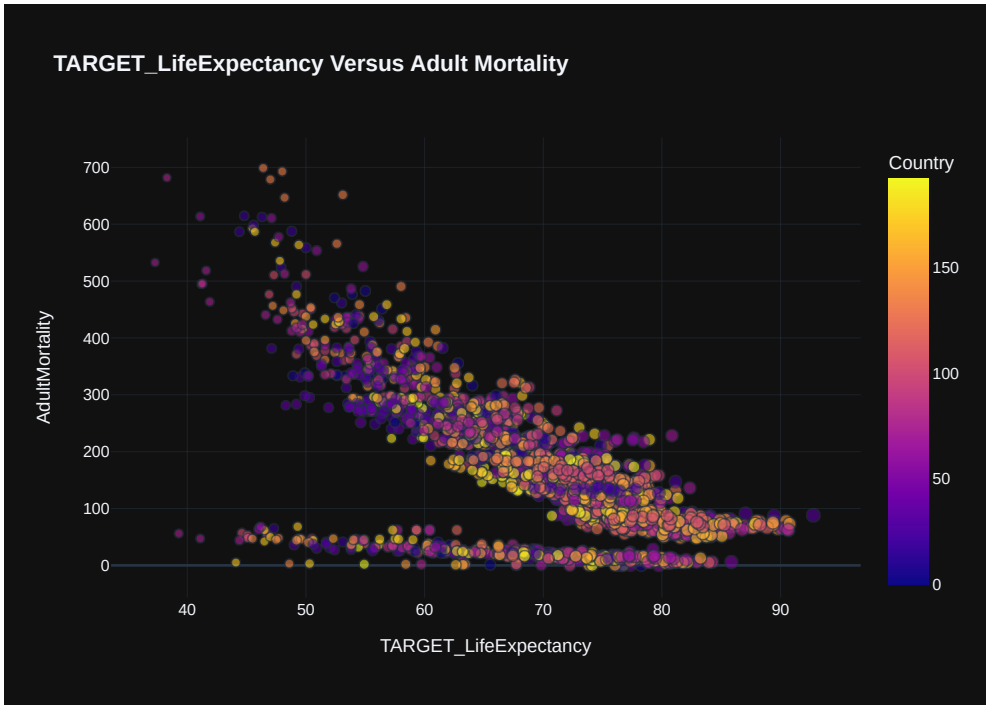


In [15…
```
fig = px.violin(lifeExpectancyFrame,x='Year',
                y='TARGET_LifeExpectancy',color='Year',
                template='plotly_dark',
                box=True,title='TARGET_LifeExpectancy Based on Year')
fig.show()
```



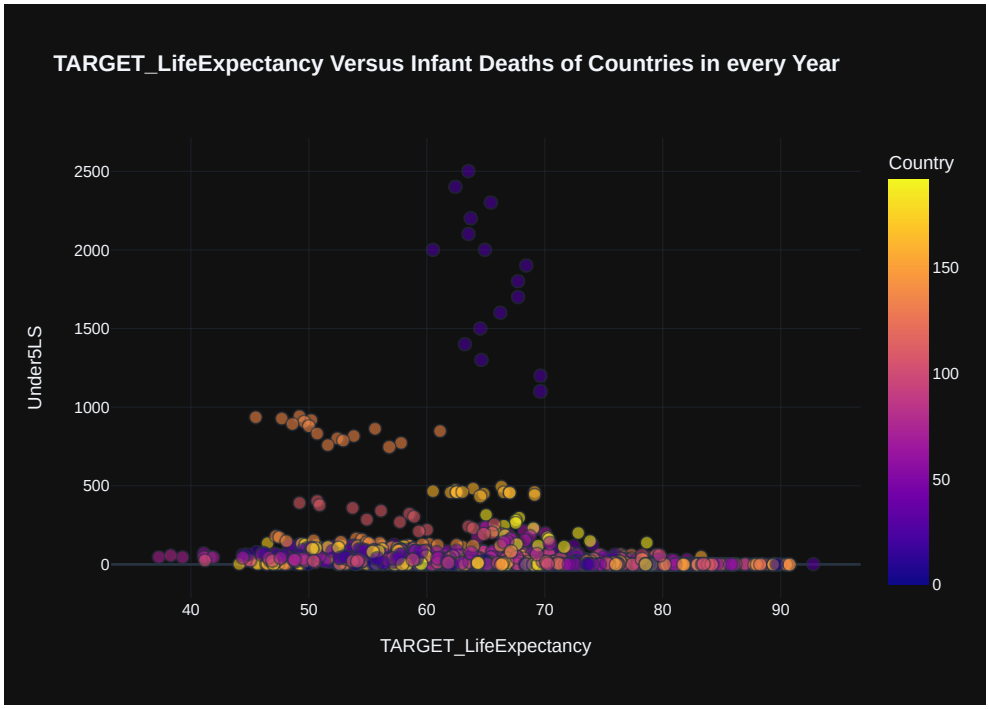
In [16…
```
fig = px.scatter(lifeExpectancyFrame, y='AdultMortality', x='TARGET_LifeExpectancy',
                 color='Country', size='TARGET_LifeExpectancy',
                 size_max=10,
                 template='plotly_dark', opacity=0.6,
                 title='<b>TARGET_LifeExpectancy Versus Adult Mortality</b>')
fig.show()
```

Loading [MathJax]/extensions/Safe.js



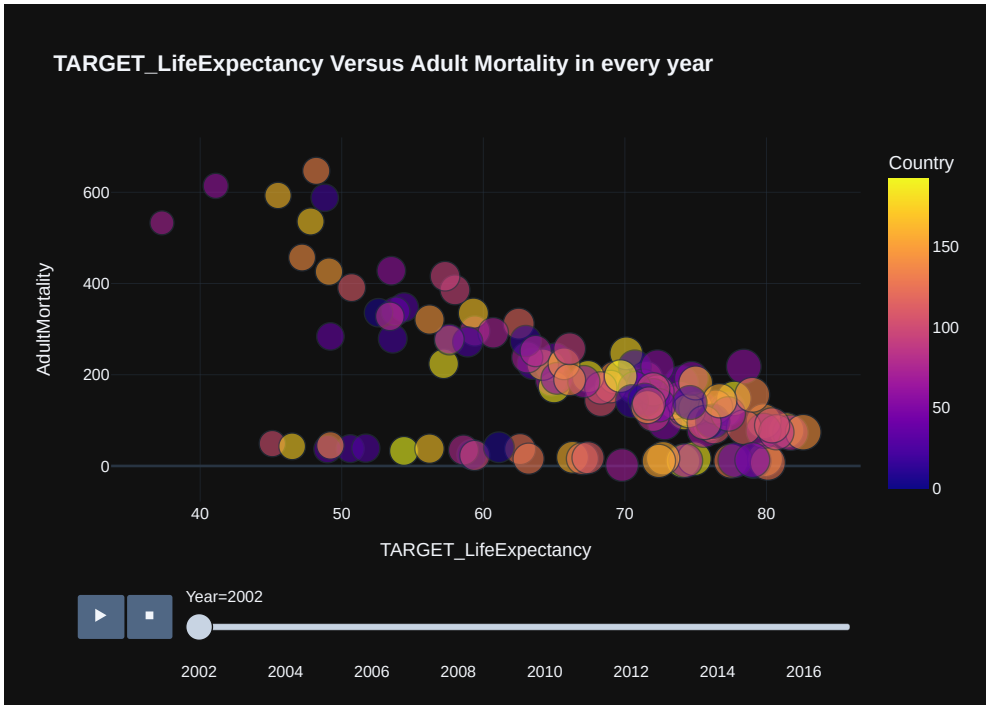
In [17…

```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'), y='Under5LS', x='TARGET_LifeExpectancy',
                 size='Year', color='Country', opacity=0.6,
                 size_max=10,
                 template='plotly_dark',
                 title='<b>TARGET_LifeExpectancy Versus Infant Deaths of Countries in every Year</b>')

fig.show()
```

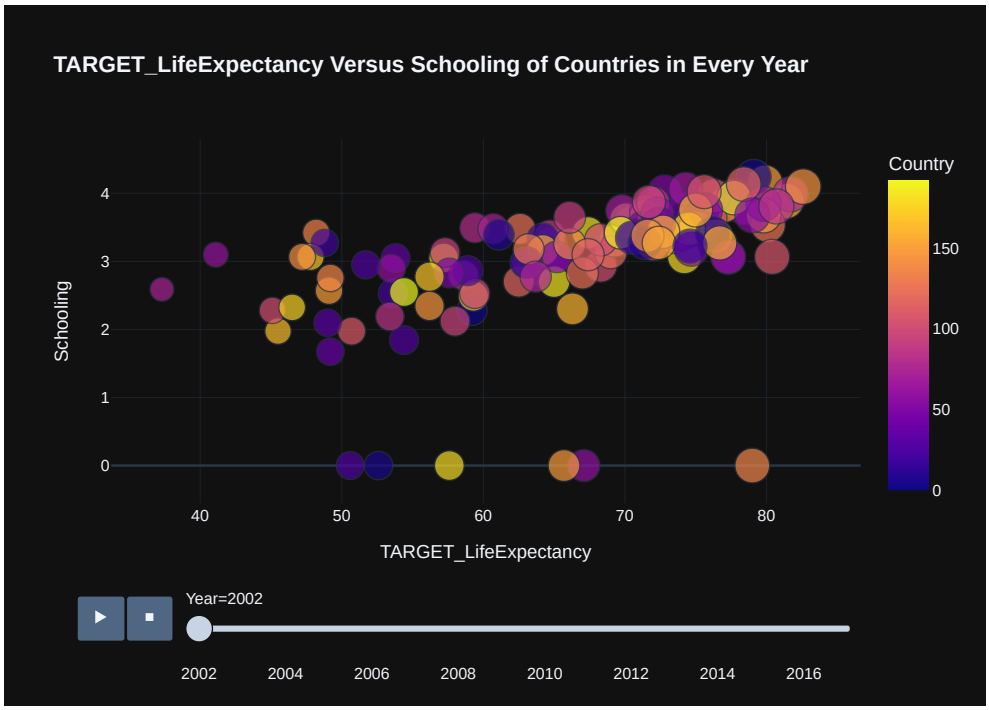


In [18…

```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'),y='AdultMortality',x='TARGET_LifeExpectancy',
                 animation_frame='Year',animation_group='Country',color='Country',
                 size='TARGET_LifeExpectancy',opacity=0.6,
                 template='plotly_dark',
                 title='<b>TARGET_LifeExpectancy Versus Adult Mortality in every year')

fig.show()
```

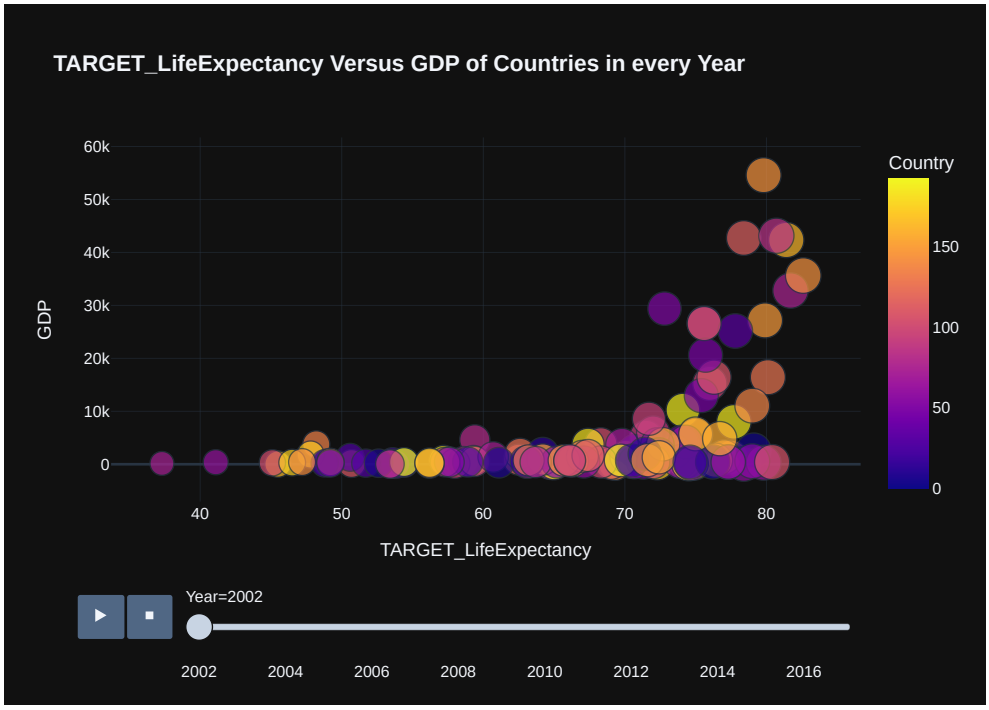


In [19…
```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'), x='TARGET_LifeExpectancy', y='Schooling',
                 animation_frame='Year', animation_group='Country',
                 color='Country', size='TARGET_LifeExpectancy',
                 template='plotly_dark',
                 title='<b>TARGET_LifeExpectancy Versus Schooling of Countries in Every Year</b>')
fig.show()
```

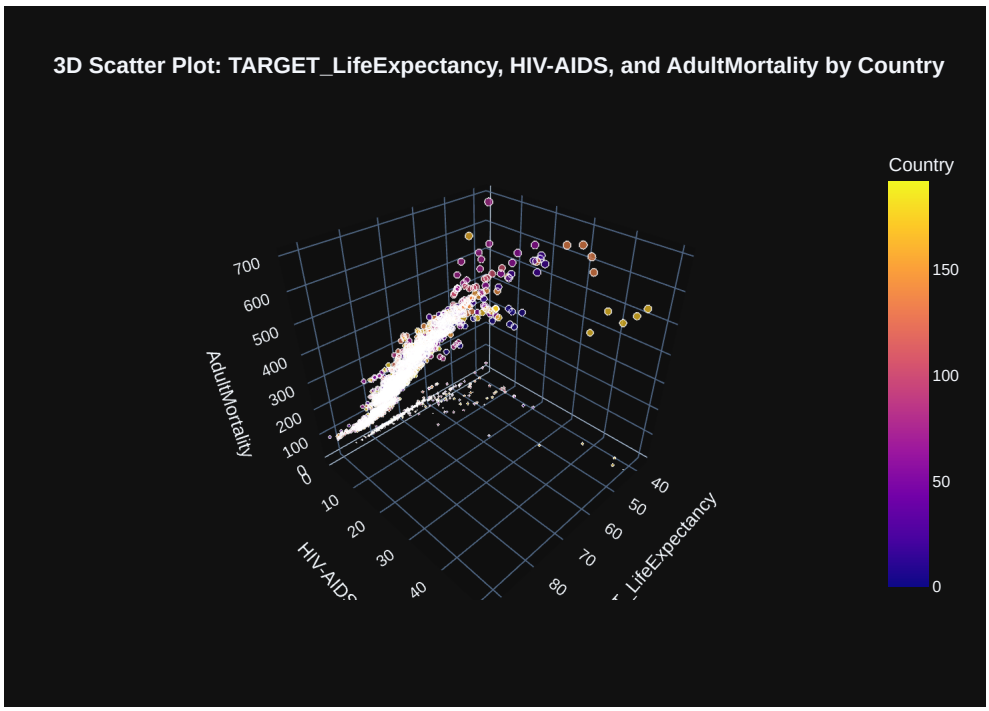


In [20…
```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'),y='GDP',x='TARGET_LifeExpectancy',animation_frame='Y
                 template='plotly_dark',color='Country',size='TARGET_LifeExpectancy',
                 title='<b>TARGET_LifeExpectancy Versus GDP of Countries in every Year')
fig.show()
```

Loading [MathJax]/extensions/Safe.js

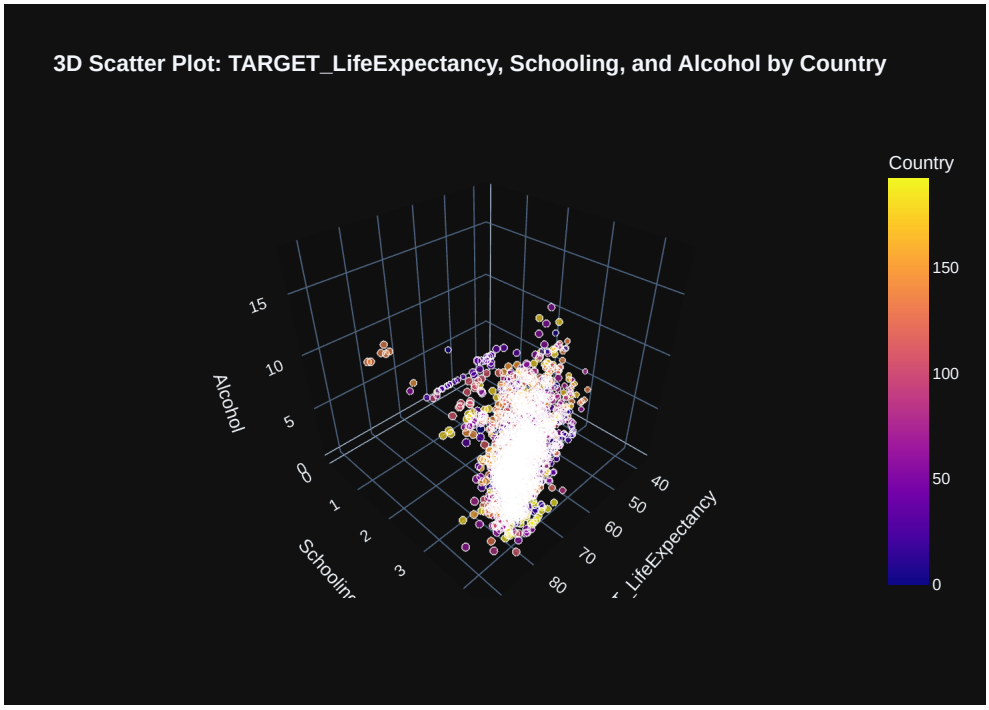


In [21…
```
fig = px.scatter_3d(lifeExpectancyFrame.sort_values(by='Year'), x='TARGET_LifeExpectancy', y='HIV-AIDS', z='Adul
                    color='Country', size='AdultMortality',
                    size_max=10,
                    template='plotly_dark',
                    title='<b>3D Scatter Plot: TARGET_LifeExpectancy, HIV-AIDS, and AdultMortality by Country</b

fig.show()
```



In [22…
```
fig = px.scatter_3d(lifeExpectancyFrame.sort_values(by='Year'), x='TARGET_LifeExpectancy', y='Schooling', z='Alc
                    color='Country', size='TARGET_LifeExpectancy',
                    size_max=10,
                    template='plotly_dark',
                    title='<b>3D Scatter Plot: TARGET_LifeExpectancy, Schooling, and Alcohol by Country</b>')

fig.show()
```

Loading [MathJax]/extensions/Safe.js



What observations did you make?

✔ Observations:

Feature Relationships: We are inspecting all features for their relation to the target: TARGET_LifeExpectancy.
Years Trend: There is a slight upward trend observed over the years, indicating an improvement in Life Expectancy over time.
Status Indicator: The distinction between 'developed' and 'developing' status is a significant indicator of Life Expectancy, potentially useful for logistic regression using a sigmoid function. However, because the data is heavily skewed as there are more examples of Developing Countries this makes it less usable.
Mortality and Infant Deaths: Adult mortality rates (for both males and females) and infant deaths are obviously linked to Life Expectancy.
Alcohol Consumption: Surprisingly, there seems to be a positive correlation between alcohol consumption and Life Expectancy. This observation might be counterintuitive and warrants further investigation.
Outliers in Expenditure: PercentageExpenditure has notable outliers that might not be appropriate for the dataset. These will be explored in more detail later.
Disease-Related Trends: Clear trends are observed in data related to diseases such as Diphtheria, HIV-AIDS, and Polio, which are expected to impact Life Expectancy.
BMI and Thinness: Both BMI and Thinness exhibit trends, suggesting their influence on Life Expectancy.
Economic Factors: IncomeCompositionOfResources and GDP show trends, indicating their correlation with Life Expectancy.
Schooling: Schooling also demonstrates a trend, further supporting its role in influencing Life Expectancy.
Overall Trends: Most of the data exhibits a trend (either a negative or positive correlation) with the target Life Expectancy, except for categorical or nominal variables like Country and ID, which are not directly correlated.
There also seems to be quite a variance in the values used for each dataset, and some noticable skews that we will need to handle using scaling in order to ensure all features are considered by our model fairly.

In [23…
```
lifeExpectancyFrame.corr().head(3)
```

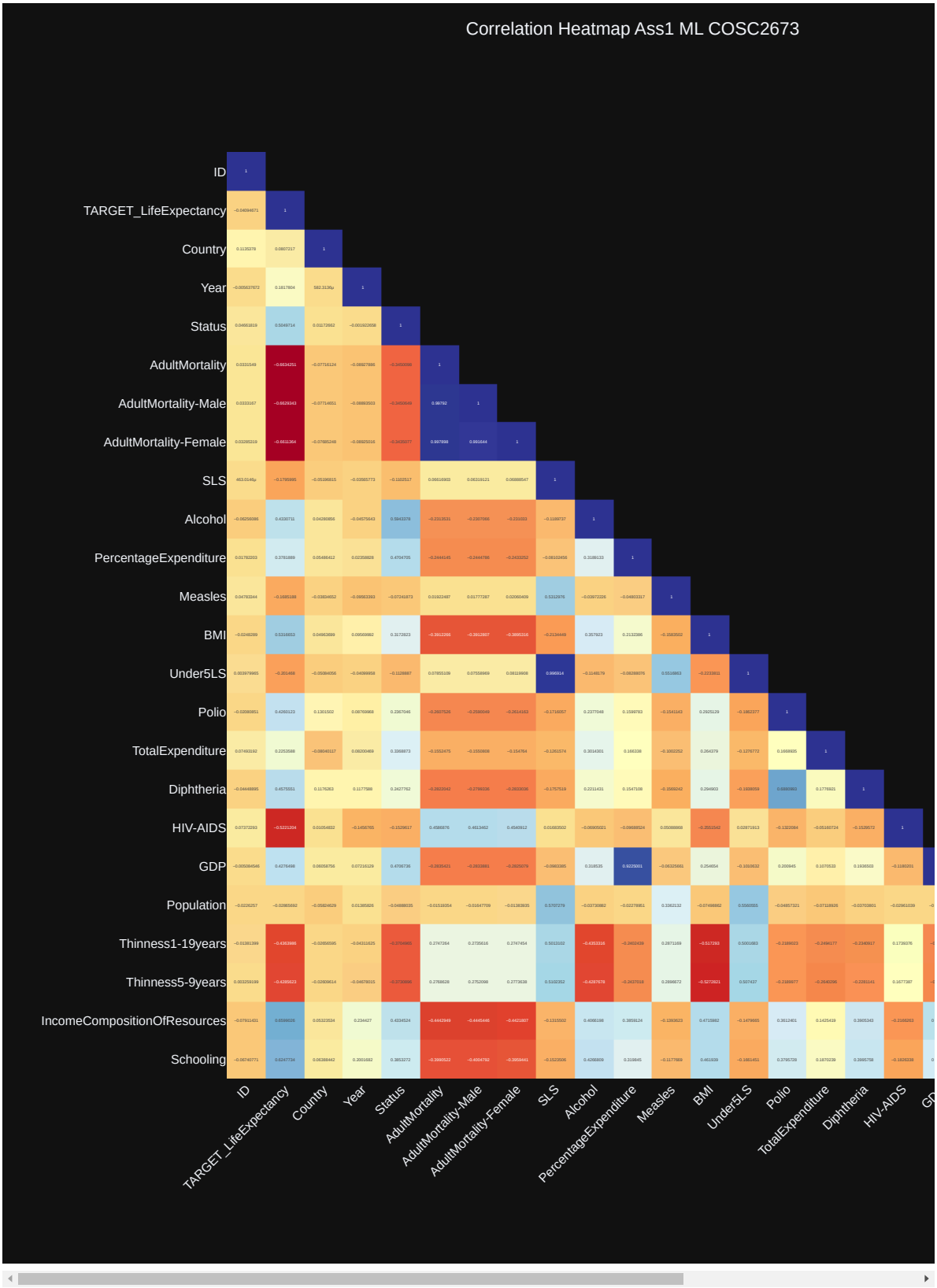
Out[23]:

	ID	TARGET_LifeExpectancy	Country	Year	Status	AdultMortality	AdultMorta
ID	1.000000	-0.040947	0.113538	-0.005638	0.046618	0.033155	0.0
TARGET_LifeExpectancy	-0.040947	1.000000	0.080722	0.181780	0.504971	-0.663425	-0.6
Country	0.113538	0.080722	1.000000	0.000582	0.011727	-0.077161	-0.0

3 rows × 24 columns

In [24…
```
# Calculate the Correlation Matrix
corr = lifeExpectancyFrame.corr()

# Manually set the upper triangle of the correlation matrix to None as not required data is symmetric :)
for i in range(len(corr)):
```

```
    for j in range(i+1, len(corr)):
        corr.iloc[i, j] = None

# Use Plotly Express to create the heatmap
fig = px.imshow(corr,
                text_auto=True,  # Display correlation values for each cell
                aspect="equal",
                color_continuous_scale='rdylbu',  # rdylbu is red (-1), yellow (0), blue (1)
                origin='upper')

# Update layout for better readability and aesthetics
fig.update_layout(
    title_text='Correlation Heatmap Ass1 ML COSC2673',
    title_x=0.5,
    width=1200,  # I found 1300 is good for 1920x1080 adjust :)
    height=1200,  # Square ^^
    autosize=False,  # use above
    xaxis_showgrid=False,  # Remove gridlines
    yaxis_showgrid=False,
    xaxis_title='',  # Remove axis titles
    yaxis_title='',
    xaxis_tickangle=-45,  # Rotate x-axis labels 45 degrees for better readability
    template='plotly_dark'  # Use a dark theme for the background
)

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



Following the analysis of our heatmap, we gain comprehensive insights that reinforce our initial observations and unveil additional relationships across the dataset:

- The heatmap effectively highlights a near 1:1 relationship among various mortality indicators, as well as between infant mortality and short lifespan indicators, underscoring their significant impact on life expectancy.
- A surprising near 1:1 correlation is observed between PercentageExpenditure and GDP, which seems counterintuitive given the expected broader influence of GDP on life expectancy. This warrants a closer examination to understand the underlying factors and potential data anomalies.
- The strength of heatmaps lies in their ability to visualize all feature relationships simultaneously, offering a holistic view beyond the pairwise correlations previously examined with the target variable. This comprehensive perspective is invaluable for identifying both expected and unexpected correlations within the dataset.
- To enhance readability and focus on unique information, I designed the heatmap to display only half of the data matrix, omitting the symmetrically duplicated half. This decision streamlines the visualization without sacrificing analytical value. For those who prefer a complete matrix view, minor adjustments to the plotting code can reveal the full heatmap.

- The inclusion of exact value labels within the heatmap further refines our analysis, enabling precise identification of correlation strengths and facilitating a more nuanced interpretation of the data relationships.

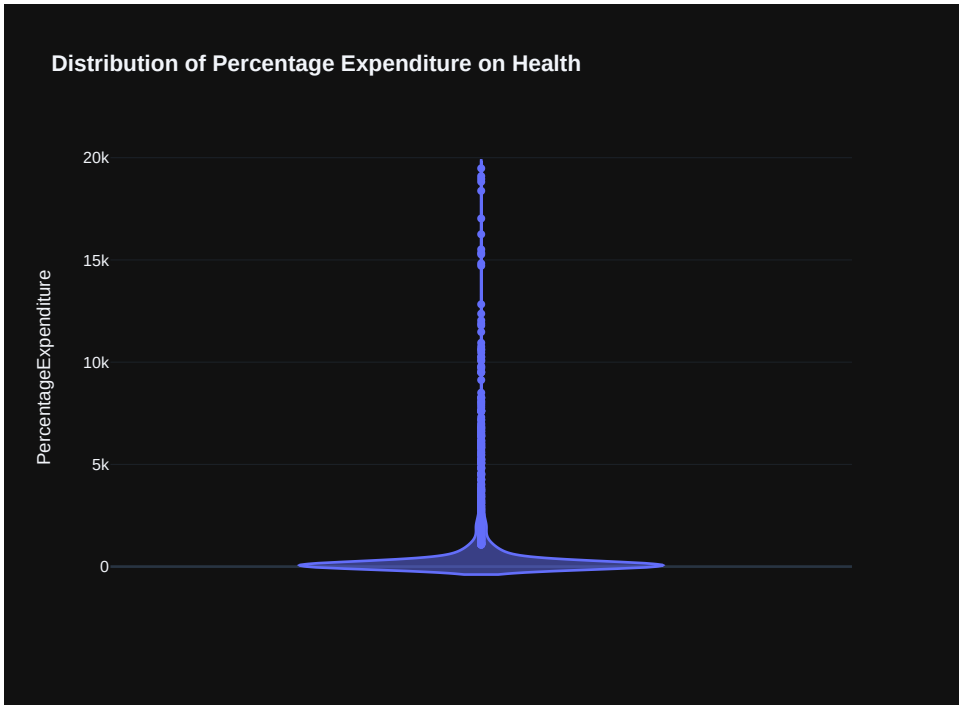
This heatmap analysis not only corroborates our earlier findings but also broadens our understanding of the intricate interplay among various features, setting the stage for deeper investigation into specific areas of interest.

2.1-1 Data Inconsistencies Identified

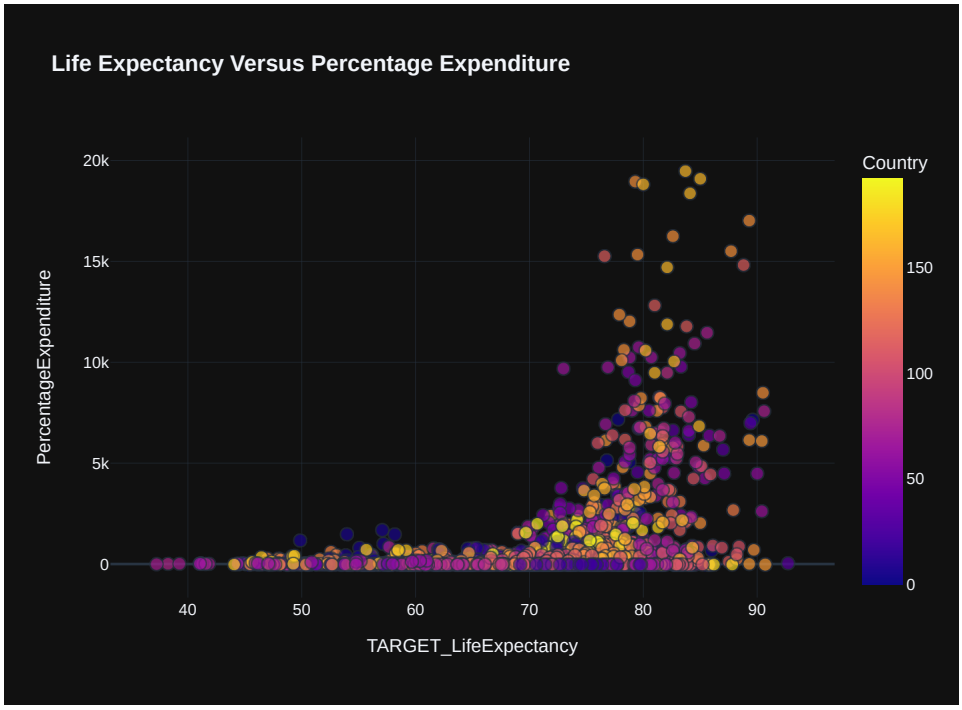
As previously identified and noted, certain aspects of our data appear to be questionable. In this section, we will delve deeper into a few of these observations, critically assess potential biases, and devise a strategic plan for mitigating these issues in our dataset moving forward.

Percentage Expenditure Column

In [25…
```
fig = px.violin(lifeExpectancyFrame, y='PercentageExpenditure',
                template='plotly_dark',
                title='<b>Distribution of Percentage Expenditure on Health</b>')

fig.show()
```



In [26…
```
fig = px.scatter(lifeExpectancyFrame, x='TARGET_LifeExpectancy', y='PercentageExpenditure',
                 color='Country', size='Year',
                 size_max=10,  # Sets the maximum marker size
                 template='plotly_dark',
                 title='<b>Life Expectancy Versus Percentage Expenditure</b>')

fig.show()
```

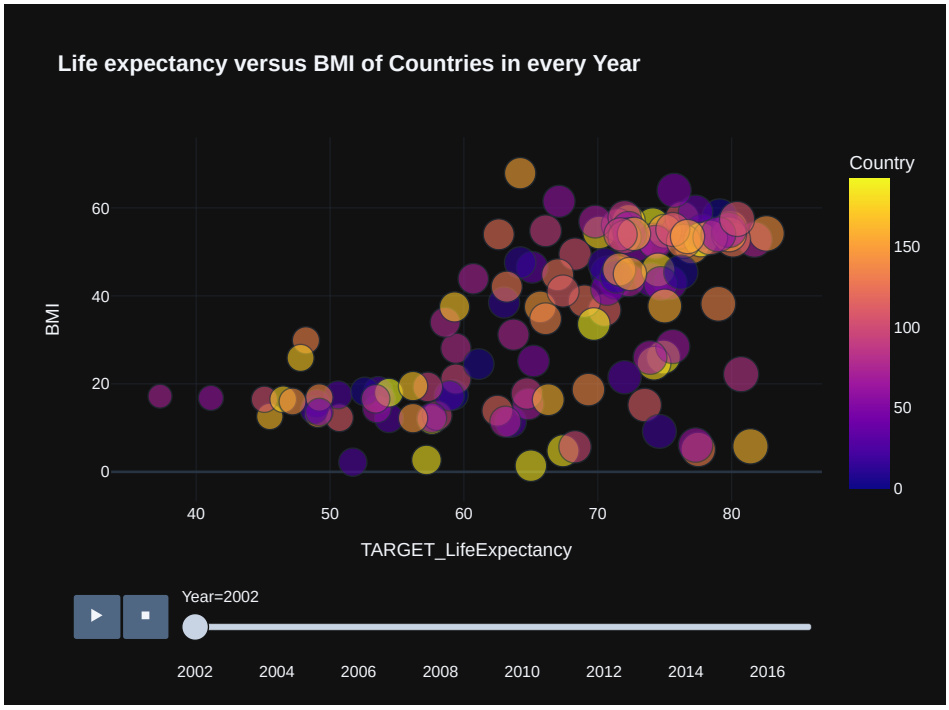


- The values for PercentageExpenditure, significantly exceed 100%, which clearly does not align with the dataset's description, which identifies this metric as the percentage of GDP allocated to health – a figure that historically has never approached levels as high as 19000%.
- While initial instincts might lean towards capping the data, removing outliers, or applying a log transformation, the exceptionally high correlation with GDP (0.93) signals a more fundamental issue.
- Further examination of the original WHO Dataset, which shares many of these values, revealed that the data in question actually represents the raw monetary amount spent on health per capita within each country.
- To rectify this, we will adjust the health expenditure figures by dividing them by the GDP per capita and subsequently multiplying by 100 to convert them into the correct percentage format.
- There are two equally plausible solutions which is to correct the metadata label so when we discuss this feature we know what the values actually are, or to as above apply the transformation to make our features independent from one another.
- Verification against authentic WHO data through a side-by-side comparison confirmed the accuracy of these adjusted PercentageExpenditure values, suggesting an oversight or mislabeling by the dataset's creator.
- ⚠ Multicollinearity, such as the observed correlation between PercentageExpenditure and GDP, can compromise the integrity of our model by causing inflated variance in coefficient estimates and obscuring the distinct influence of correlated predictors on the target variable.

BMI Column

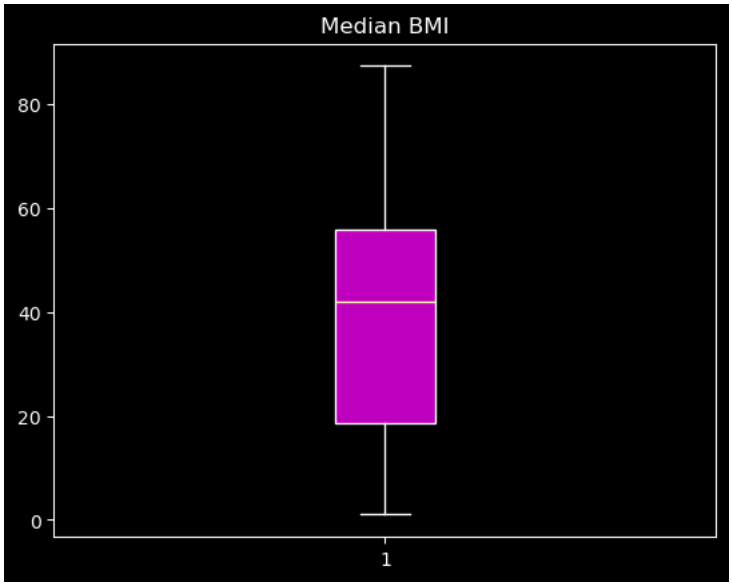
In [27…
```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'),y='BMI',x='TARGET_LifeExpectancy',animation_fram
                 template='plotly_dark',color='Country',size='TARGET_LifeExpectancy',opacity=0.6,
                 title='<b> Life expectancy versus BMI of Countries in every Year')
fig.show()
```

Loading [MathJax]/extensions/Safe.js



In [28…

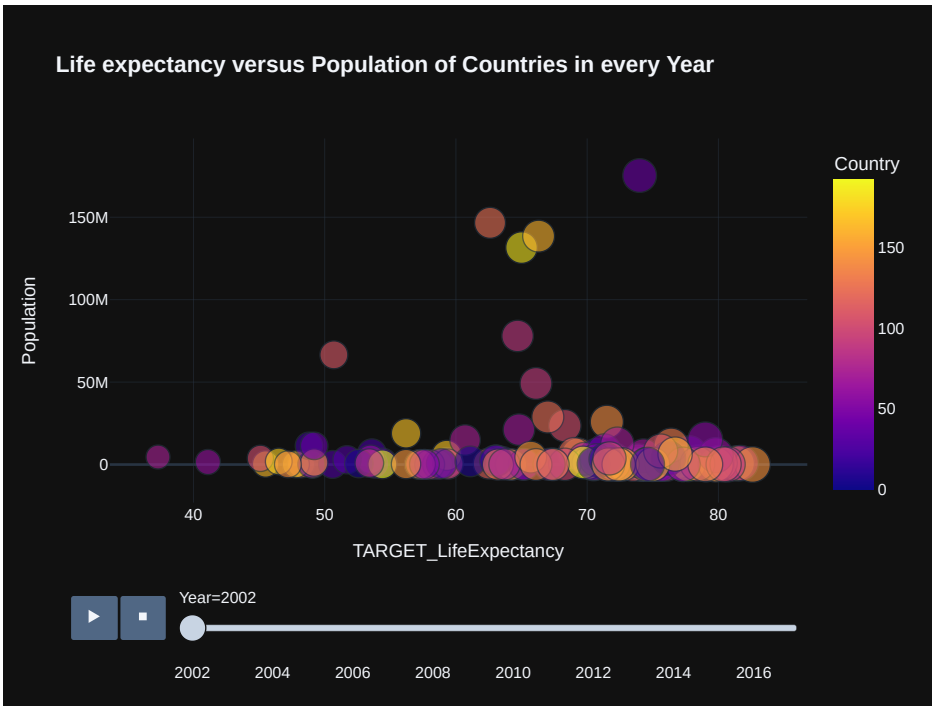
```
# Creating a boxplot
plt.boxplot(lifeExpectancyFrame['BMI'], patch_artist=True, boxprops=dict(facecolor='m', color='w'))

# Adding title and customizing the plot
plt.title('Median BMI')

plt.show()
```



- According to WHO 2015 records, the lowest BMI recorded is 20.5 (Eritrea) and the highest is 32.5 (Nauru).
- In contrast, our dataset shows an unrealistic mean BMI of ~47, which exceeds the maximum recorded real value, with extreme values ranging from nearly 0 to 90.
- Despite this, the distribution around the mean correlates with our target variable, Life Expectancy, indicating the data distribution is relatively coherent.
- To address the inflated mean BMI, I plan to use scaling techniques. This will adjust the mean to a more realistic range while keeping the distribution's shape intact. I could then apply some outlier capping, to further remedy this issue.
- Visual analyses have also uncovered biologically improbable fluctuations in BMI, such as dramatic year-over-year changes within some countries which I'm not exactly sure how to fix at this point-in-time.
- These findings highlight the importance of thorough data validation and correction, ensuring the dependability of our analyses and predictive models.

Population Column

In [29…
```
fig = px.scatter(lifeExpectancyFrame.sort_values(by='Year'),y='Population',x='TARGET_LifeExpectancy',anima
                 template='plotly_dark',color='Country',size='TARGET_LifeExpectancy',opacity=0.6,
                 title='<b> Life expectancy versus Population of Countries in every Year')
fig.show()
```



- Population is a noteworthy example within our dataset where the data is represented by significantly large numbers.
- To prevent the Population feature from unduly influencing our model due to its large scale.
- I plan to minmax scaling techniques during the Data Preprocessing phase to preserve the distribution, but make it only from 0 - 1.
- It is also a good approach to perhaps normalize Population values, as it is skewed.

- We will address these concerns, and many other identified outliers we identified earlier in our EDA during the Data Preprocessing phase.
- Our approach will include employing scaling, transformation, and normalization techniques to ensure data uniformity across features.
- To mitigate issues like overfitting and multicollinearity, we'll integrate regularization methods into our model training process.
- For outlier identification and removal, we will leverage robust statistical methods, like:
  - The Interquartile Range (IQR) method, which is my preferred approach due to its effectiveness in handling skewed data distributions.
- These targeted strategies for outlier management are aimed at refining our dataset, thus paving the way for achieving optimal model performance.

2.2 Data Splitting

- In this step, I will split `training.csv` into training and validation subsets to enable the training and testing of our model before we evaluate it against unseen 'blind' data that simulates a real input.

- Following the recommendation by Azadeh and aligning with my personal preference for datasets of this size, I will implement an 80/20 split.
- This approach is fundamental to our model development process, allowing us to assess the model's ability to generalize well beyond the training data.
- A key aspect we emphasized in our lecture on supervised learning is the importance of having independent and identically distributed test data to avoid data leakage and ensure an unbiased evaluation of the final model.
- Given that we possess a distinct `test.csv`, which simulates real-world 'blind' data by excluding the target variable, our focus for the moment will shift towards dividing `training.csv` file into training and validation sets.
- The division of `training.csv` is crucial for both the training and subsequent internal evaluation of our models, allowing us to refine them prior to the final assessment against the 'blind' test data.
- To achieve a balanced distribution, I prefer an 80/20 split between the training and validation subsets, respectively, which suits datasets of this scale.
- This structured approach underscores the model development phase, enabling us to meticulously gauge the model's generalization capabilities and ensure robustness before confronting the unseen data in `test.csv`.

In [30…
```
lifeExpectancy_X = lifeExpectancyFrame.drop(['TARGET_LifeExpectancy', 'ID'], axis=1)
lifeExpectancy_Y = lifeExpectancyFrame['TARGET_LifeExpectancy']
```

In [31…
```
from sklearn.model_selection import train_test_split

with pd.option_context('mode.chained_assignment', None):
    lifeExpectancy_X_train, lifeExpectancy_X_validation, lifeExpectancy_Y_train, lifeExpectancy_Y_validat
```

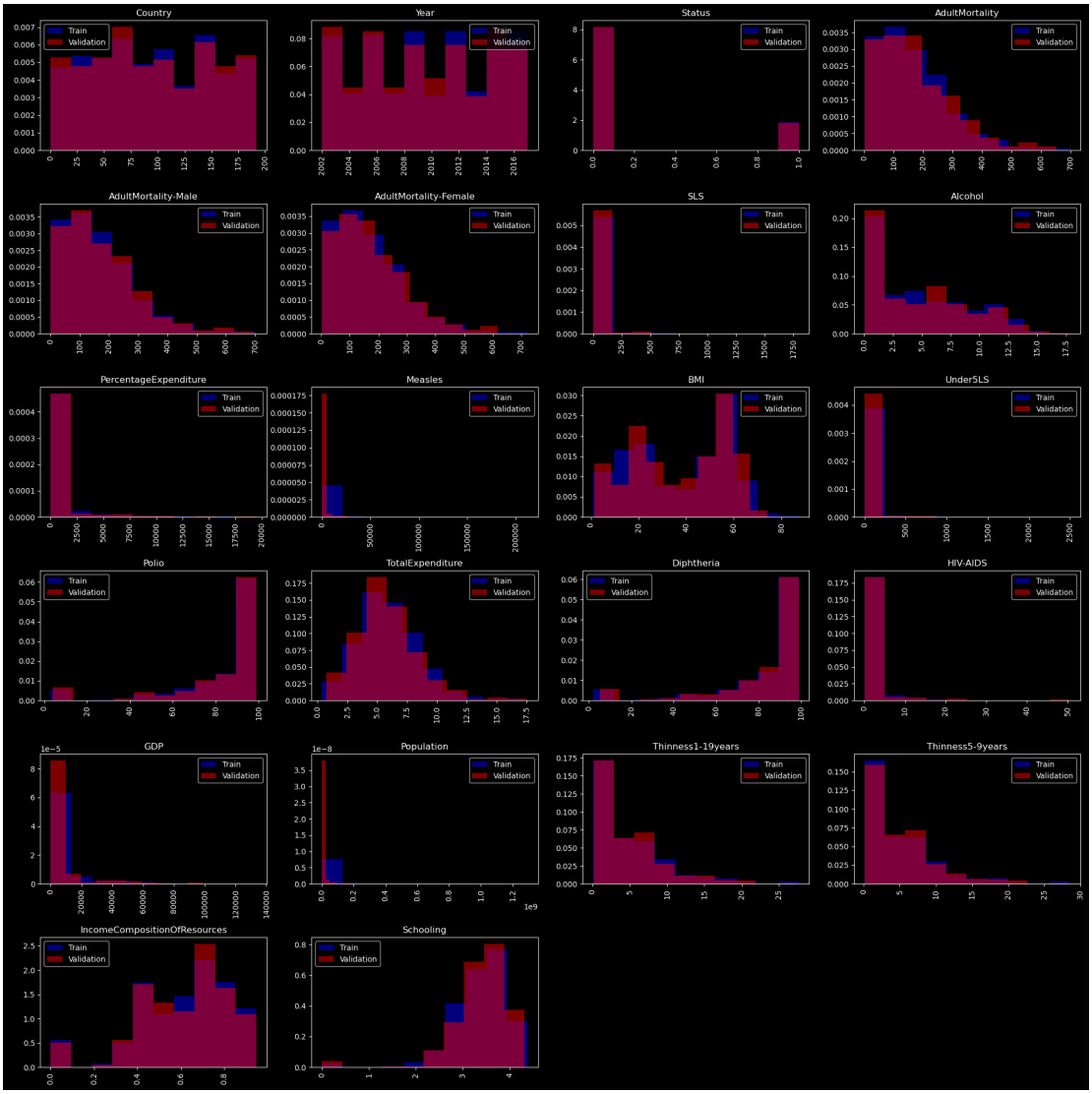
In [32…
```
print("Number of instances in the original dataset is {}. After splitting, Train has {} instances and Val
      .format(lifeExpectancyFrame.shape[0],
              lifeExpectancy_X_train.shape[0],
              lifeExpectancy_X_validation.shape[0],
              lifeExpectancy_X_train.shape[0] + lifeExpectancy_X_validation.shape[0]))
```

```
Number of instances in the original dataset is 2071. After splitting, Train has 1656 instances and Validat
ion has 415 instances. The sum of Train and Validation instances is 2071, which matches the original datas
et.
```

2.2-1 Checking for Data Leaks

- Random splitting is essential but may lead to leakage if two splits are not truly independent.
- We will utilize insights from Exploratory Data Analysis (EDA) to detect any hidden sources of leakage in the dataset.
- We will examine histograms for each attribute in the training and validation sets, using different colors, to ensure the splits are identically distributed.

In [33…
```
# Assuming lifeExpectancy_X_train and lifeExpectancy_X_test are your training and testing datasets, resp

plt.figure(figsize=(20, 20))

# Loop through the columns of the training frame
for i, col in enumerate(lifeExpectancy_X_train.columns):
    plt.subplot(6, 4, i + 1)  # Adjust the subplot grid size as needed
    # Plot the histogram for the train dataset column
    plt.hist(lifeExpectancy_X_train[col], alpha=0.5, color='b', density=True, label='Train')
    # Plot the histogram for the test dataset column
    plt.hist(lifeExpectancy_X_validation[col], alpha=0.5, color='r', density=True, label='Validation')
    plt.title(col)
    plt.xticks(rotation='vertical')
    plt.legend()

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



Key Observations:

✔ The distributions of attributes within the training set closely align with those in the test set, indicating that the random splitting process has effectively preserved the dataset's overall statistical properties, meaning we can move on.

2.3 Data Preprocessing

⚠ Critical Reminder: Consistency in Data Preprocessing

It is crucial to apply the same preprocessing steps to all of our datasets; training, validation, and the blind test dataset. This ensures uniformity across all data, maintaining the integrity of our predictions.
Failing to do so may result in predictions based on altered or unaccounted values, such as outliers, potentially skewing our model's performance.

In [34…

```
# Copy the original training and testing sets for scaling
lifeExpectancy_X_train_scaled = lifeExpectancy_X_train.copy()
lifeExpectancy_X_validation_scaled = lifeExpectancy_X_validation.copy()
```

⚠ Warning: Please run only once.

In [35…
```
# Correcting the PercentageExpenditure values in the scaled datasets
lifeExpectancy_X_train_scaled['PercentageExpenditure'] = (lifeExpectancy_X_train_scaled['PercentageExpe
lifeExpectancy_X_validation_scaled['PercentageExpenditure'] = (lifeExpectancy_X_validation_scaled['Perc

# Descriptive statistics for the corrected PercentageExpenditure in the scaled training set
desc_pe_scaled_train = lifeExpectancy_X_train_scaled[['PercentageExpenditure']].describe()
print("Scaled Training Dataset - Corrected PercentageExpenditure:\n", desc_pe_scaled_train)

# Descriptive statistics for the corrected PercentageExpenditure in the scaled test set
desc_pe_scaled_validation = lifeExpectancy_X_validation_scaled[['PercentageExpenditure']].describe()
print("\nScaled Validation Dataset - Corrected PercentageExpenditure:\n", desc_pe_scaled_validation)
```

```
Scaled Training Dataset - Corrected PercentageExpenditure:
       PercentageExpenditure
count            1656.000000
mean                7.263274
std                 5.385992
min                 0.000000
25%                 1.580356
50%                 7.388397
75%                11.535716
max                25.883941

Scaled Validation Dataset - Corrected PercentageExpenditure:
       PercentageExpenditure
count             415.000000
mean                7.312810
std                 5.563964
min                 0.000000
25%                 1.401786
50%                 7.607252
75%                11.820955
max                22.928571
```

- Adjusting the values for PercentageExpenditure resulted in an average reduction of 0.01 in the R^2 score, indicating that running the aforementioned cell may not lead to optimal performance.
- This suggests that the corrected PercentageExpenditure has little to no correlation with LifeExpectancy.
- Conversely, adjustments related to GDP led to a slight improvement in model performance, implying a more significant correlation between GDP and LifeExpectancy.

- 0.01 R^2 score increase is noted from below.
- Although inaccuracies in BMI data were noted, a correlation was observed in the heatmap between thinness and characteristics indicative of wealthier, less food-scarce nations which we can expect to be correlated to BMI.
- The distribution of BMI values exhibits a roughly Gaussian shape but include notable outliers.
- To correct for this, we plan to adjust the dataset by shifting all BMI values downwards, aiming for a lower mean and upper limit, more aligned with realistic country-specific BMI records.

In [36…
```
# Create a copy of the DataFrame for the version before outlier adjustment so we can plot the differenc
lifeExpectancy_X_train_scaled_before_outlier_adjustment = lifeExpectancy_X_train_scaled.copy()
lifeExpectancy_X_validation_scaled_before_outlier_adjustment = lifeExpectancy_X_validation_scaled.copy(

# Function to filter out outliers
def filter_outliers(df):
    for col in df.columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# Apply the outlier filtering function to both the training and validation datasets
lifeExpectancy_X_train_scaled = filter_outliers(lifeExpectancy_X_train_scaled)
lifeExpectancy_X_validation_scaled = filter_outliers(lifeExpectancy_X_validation_scaled)
```

In [37…
```
from sklearn.preprocessing import PowerTransformer

# Assuming lifeExpectancy_X_train and lifeExpectancy_X_test are your initial datasets

# Define attributes for normalization
Norm_attributes = ['SLS', 'HIV-AIDS', 'AdultMortality', 'AdultMortality-Male', 'Schooling',
                   'Thinness5-9years', 'Polio', 'Diphtheria', 'Alcohol', 'Under5LS', 'IncomeComposit
```

```
# Initialize PowerTransformer
powertransformer = PowerTransformer(method='yeo-johnson', standardize=False)

# Transforming the specified attributes in the training set
lifeExpectancy_X_train_scaled = lifeExpectancy_X_train.copy()  # Creating a scaled version of the trai
lifeExpectancy_X_train_scaled[Norm_attributes] = powertransformer.fit_transform(lifeExpectancy_X_train[

# Transforming the specified attributes in the test set
lifeExpectancy_X_validation_scaled = lifeExpectancy_X_validation.copy()  # Creating a scaled version o
lifeExpectancy_X_validation_scaled[Norm_attributes] = powertransformer.transform(lifeExpectancy_X_valid
```

In [38…

```
from sklearn.preprocessing import MinMaxScaler

# Initialize MinMaxScaler
minmax_scaler = MinMaxScaler()

# Transforming all features in the training set
lifeExpectancy_X_train_scaled = minmax_scaler.fit_transform(lifeExpectancy_X_train_scaled)

# Transforming all features in the test set
lifeExpectancy_X_validation_scaled = minmax_scaler.transform(lifeExpectancy_X_validation_scaled)

# Convert the scaled arrays back to DataFrames to retain column names (if necessary)
lifeExpectancy_X_train_scaled = pd.DataFrame(lifeExpectancy_X_train_scaled, columns=lifeExpectancy_X_t
lifeExpectancy_X_validation_scaled = pd.DataFrame(lifeExpectancy_X_validation_scaled, columns=lifeExpec
```

In [39…

```
# Temporarily adjust the Pandas display options within the context manager
with pd.option_context('display.max_columns', None):  # None means unlimited
    display(lifeExpectancy_X_train_scaled.describe())

# Temporarily adjust the Pandas display options within the context manager
with pd.option_context('display.max_columns', None):  # None means unlimited
    display(lifeExpectancy_X_validation_scaled.describe())
```

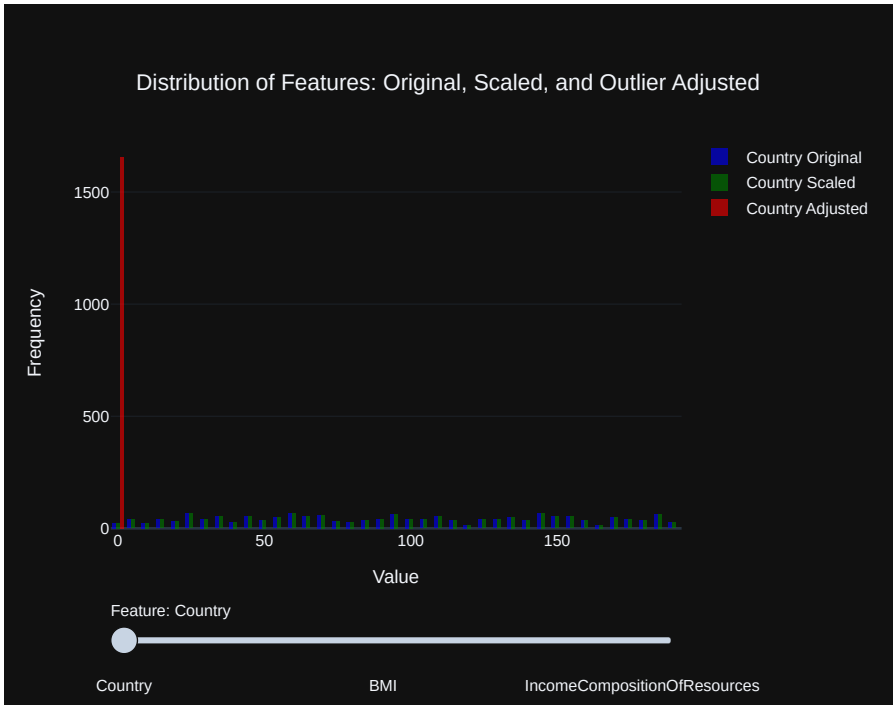
	Country	Year	Status	AdultMortality	AdultMortality-Male	AdultMortality-Female	SL
count	1656.000000	1656.000000	1656.000000	1656.000000	1656.000000	1656.000000	1656.00000
mean	0.496468	0.504469	0.186594	0.442304	0.447810	0.222060	0.35783
std	0.285370	0.307475	0.389703	0.192211	0.189643	0.163046	0.29038
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.250000	0.266667	0.000000	0.320748	0.330007	0.100000	0.00000
50%	0.489583	0.533333	0.000000	0.455354	0.460904	0.197222	0.33530
75%	0.750000	0.800000	0.000000	0.573923	0.579737	0.313889	0.62903
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

	Country	Year	Status	AdultMortality	AdultMortality-Male	AdultMortality-Female	SLS	
count	415.000000	415.000000	415.000000	415.000000	415.000000	415.000000	415.000000	41
mean	0.497465	0.488353	0.180723	0.455020	0.460077	0.235067	0.365460	
std	0.287543	0.308183	0.385253	0.197451	0.194367	0.172390	0.281769	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.265625	0.200000	0.000000	0.323112	0.338023	0.104167	0.000000	
50%	0.484375	0.466667	0.000000	0.460277	0.464129	0.202778	0.335307	
75%	0.750000	0.733333	0.000000	0.592841	0.600631	0.326389	0.622963	
max	1.000000	1.000000	1.000000	0.967199	0.994617	0.855556	0.997189	

In [40…

```
import plotly.graph_objects as go

# Create a figure with subplots
fig = go.Figure()

# List of all features in the dataset
all_features = lifeExpectancy_X_train.columns.tolist()

# Add traces for each state of each feature
for feature in all_features:
    # Original data
    fig.add_trace(
        go.Histogram(x=lifeExpectancy_X_train[feature], name=f'{feature} Original',
                     marker_color='blue', opacity=0.6, visible=False)
    )

    # Scaled data before outlier adjustment
    fig.add_trace(
```

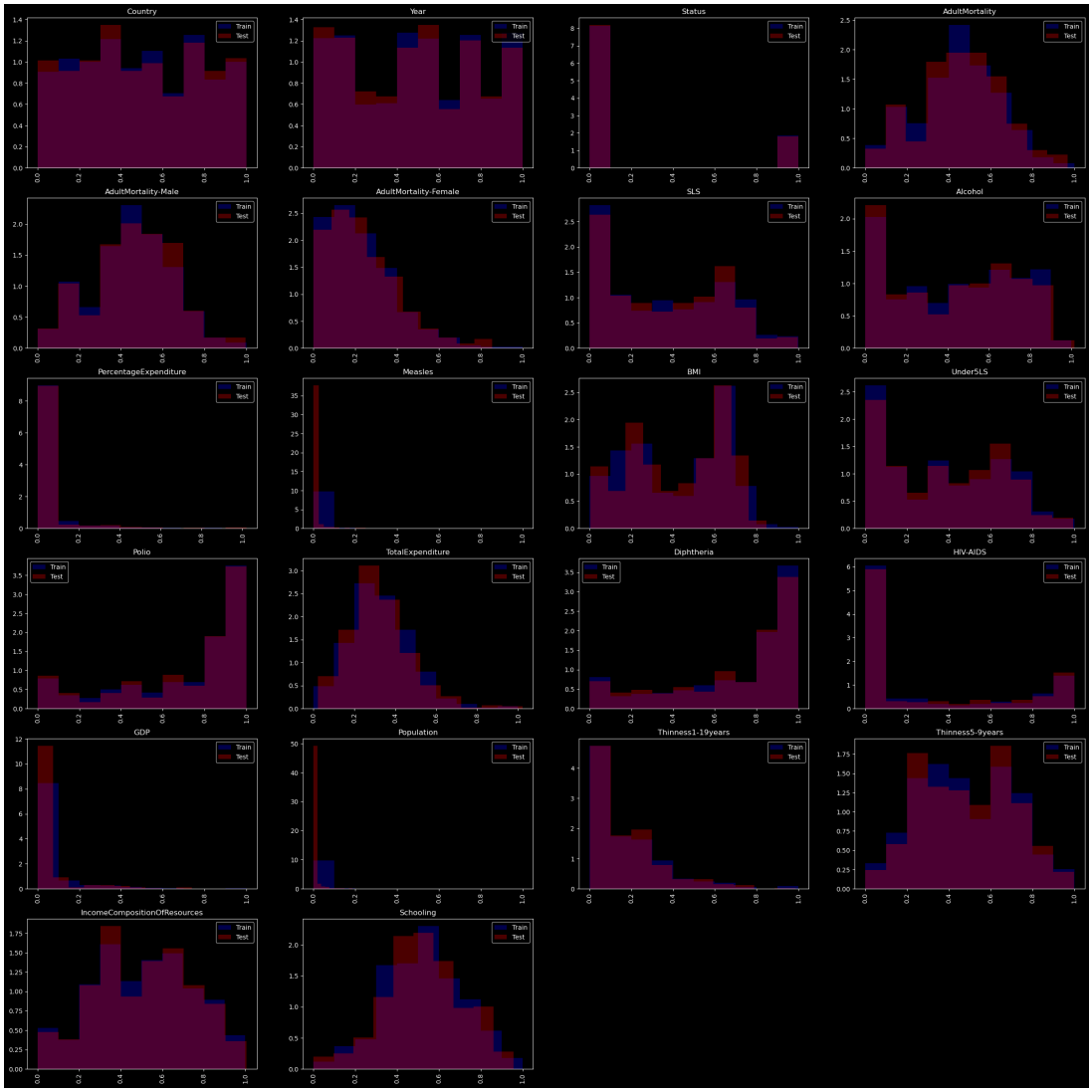
```
        go.Histogram(x=lifeExpectancy_X_train_scaled_before_outlier_adjustment[feature], name=f'{featu
                     marker_color='green', opacity=0.6, visible=False)
    )

    # Scaled data after outlier adjustment
    fig.add_trace(
        go.Histogram(x=lifeExpectancy_X_train_scaled[feature], name=f'{feature} Adjusted',
                     marker_color='red', opacity=0.6, visible=False)
    )

# Create the slider steps
steps = []
for i, feature in enumerate(all_features):
    step = dict(
        method='update',
        args=[{'visible': [False] * len(fig.data)},
              {'title': f'Distribution of {feature}: Original, Scaled, and Outlier Adjusted'}],
        label=feature
    )
    # Toggle i-th, (i+1)-th, and (i+2)-th traces to "True" to make them visible
    step['args'][0]['visible'][3*i:3*i+3] = [True, True, True]
    steps.append(step)

# Create and add the slider
sliders = [dict(
    active=0,
    currentvalue={"prefix": "Feature: "},
    pad={"t": 50},
    steps=steps
)]

fig.update_layout(
    sliders=sliders,
    title={'text': f'Distribution of Features: Original, Scaled, and Outlier Adjusted', 'y':0.9, 'x':0.
    xaxis_title="Value",
    yaxis_title="Frequency",
    bargap=0.2,
    template='plotly_dark'
)

# Show the first feature by default
for i in range(3):
    fig.data[i].visible = True

fig.show()
```



In [41…

```
plt.figure(figsize=(30,30))
for i, col in enumerate(lifeExpectancy_X_train_scaled.columns):
    plt.subplot(6,4,i+1)
    plt.hist(lifeExpectancy_X_train_scaled[col], alpha=0.3, color='b', density=True, label='Train')
    plt.hist(lifeExpectancy_X_validation_scaled[col], alpha=0.3, color='r', density=True, label='Test'
    plt.title(col)
    plt.xticks(rotation='vertical')
    plt.legend()
```

Loading [MathJax]/extensions/Safe.js



3.0 Baseline Multivariate Linear Regression Model

- The methodology adopted for this project involved starting with a multivariate regression model that included all features, (and in my opinion no pre-processing) serving as the Baseline to gauge feature correlations before proceeding with feature transformation, regularization, feature selection, hyperparameter tuning, and model optimization.
- This approach aligns with the assignment's explicit requirement aimed at enhancing fundamental ML skills, emphasizing a comprehensive understanding before delving into more advanced topics.
- Incorporating methodologies akin to those encountered in lab exercises, the project showcases the impact of preprocessing steps through side-by-side comparisons, illustrating the scaling and transformation effects on the data.
- While various metrics were considered to evaluate model performance, particular emphasis was placed on RMSE and R^2. These metrics are favored due to me being more comfortable with them, having a solid grasp of their implications, what constitutes a good score, and their operational principles.
- **Metrics Overview:** RMSE (Root Mean Square Error) measures the model's prediction error magnitude, while R^2 (Coefficient of Determination) assesses the proportion of the variance in the dependent variable that is predictable from the independent variables. I also included Mean Absolute Error (MAE)

In [42…

```
from sklearn.linear_model import LinearRegression

baseline_lr = LinearRegression().fit(lifeExpectancy_X_train_scaled, lifeExpectancy_Y_train)

print("Parameter of the Linear model (scaled): ", baseline_lr.coef_)
print("Intercept of the Linear model (scaled): ", baseline_lr.intercept_)

lifeExpectancy_Y_validation_pred = baseline_lr.predict(lifeExpectancy_X_validation_scaled)
```

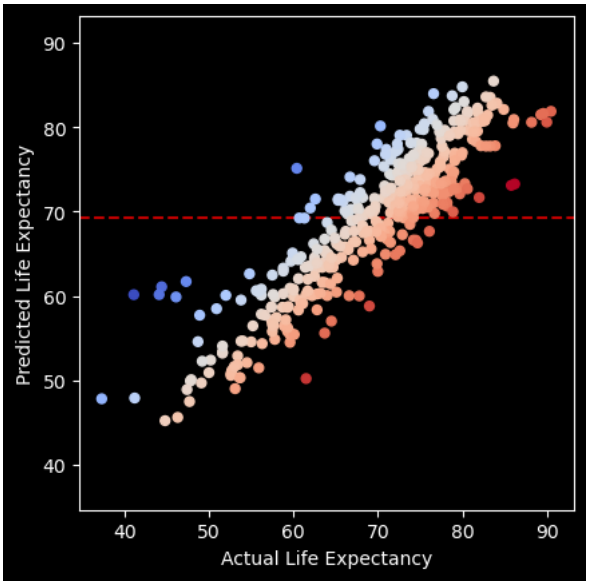
```
Parameter of the Linear model (scaled):  [ -0.39378417   1.86544693   1.86399912  81.30018999 -57.1011
5669
 -42.10795088  10.68992381   2.32100919   3.17305247  -6.92949773
  -0.65340318 -13.29337132   0.85392526  -1.28497117   2.74261616
  -6.83538427  -0.32232595   1.18171006   2.23882838  -3.71043702
   7.26541666   3.15811452]
Intercept of the Linear model (scaled):  63.176196708576796
```

In [43…
```
# Train the linear regression model without feature scaling
baseline_lr_us = LinearRegression().fit(lifeExpectancy_X_train, lifeExpectancy_Y_train)
lifeExpectancy_Y_validation_us_pred = baseline_lr_us.predict(lifeExpectancy_X_validation)

# Train the linear regression model with feature scaling (assuming lifeExpectancy_X_validation_scaled
baseline_lr = LinearRegression().fit(lifeExpectancy_X_train_scaled, lifeExpectancy_Y_train)
lifeExpectancy_Y_validation_pred = baseline_lr.predict(lifeExpectancy_X_validation_scaled)
```

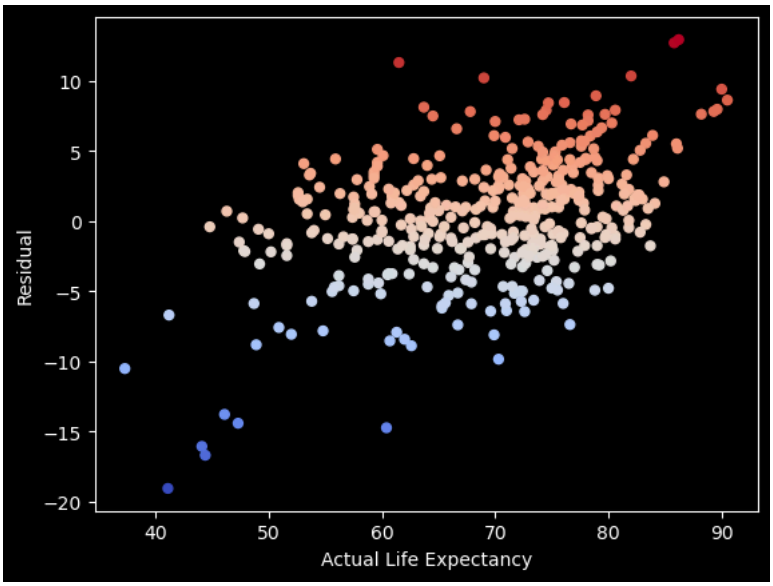
In [44…
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Calculate R² scores
r2_us_lr = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_us_pred)
r2_lr = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred)

# Calculate RMSE
rmse_us_lr = mean_squared_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_us_pred, squ
rmse_lr = mean_squared_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred, squared=F

# Calculate MAE
mae_us_lr = mean_absolute_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_us_pred)
mae_lr = mean_absolute_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred)

# Print R² scores
print('The R² score for the linear regression model (without feature scaling) is: {:.3f}'.format(r2_u
print('The R² score for the linear regression model (with feature scaling) is: {:.3f}'.format(r2_lr))

# Print RMSE
print('The RMSE for the linear regression model (without feature scaling) is: {:.3f}'.format(rmse_us_
print('The RMSE for the linear regression model (with feature scaling) is: {:.3f}'.format(rmse_lr))

# Print MAE
print('The MAE for the linear regression model (without feature scaling) is: {:.3f}'.format(mae_us_lr
print('The MAE for the linear regression model (with feature scaling) is: {:.3f}'.format(mae_lr))
```

```
The R² score for the linear regression model (without feature scaling) is: 0.754
The R² score for the linear regression model (with feature scaling) is: 0.797
The RMSE for the linear regression model (without feature scaling) is: 4.797
The RMSE for the linear regression model (with feature scaling) is: 4.352
The MAE for the linear regression model (without feature scaling) is: 3.585
The MAE for the linear regression model (with feature scaling) is: 3.270
```

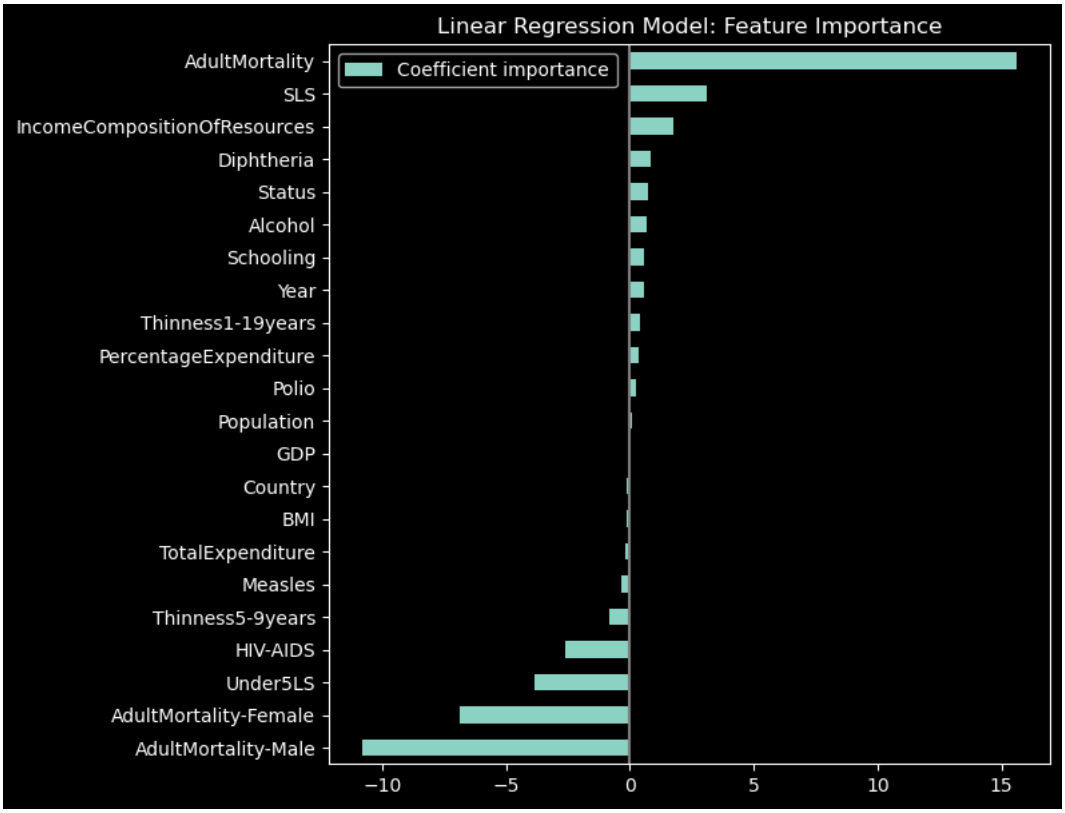
In [45…
```
fig, ax = plt.subplots()
errors = lifeExpectancy_Y_validation - lifeExpectancy_Y_validation_pred
ax.scatter(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred, c=errors, s=25, cmap=plt.cm

lims = [
    np.min([ax.get_xlim(), ax.get_ylim()]),  # min of both axes
    np.max([ax.get_xlim(), ax.get_ylim()]),  # max of both axes
]

ax.plot(lims, lims, 'k--', alpha=0.75, zorder=0)
ax.plot(lims, [np.mean(lifeExpectancy_Y_train),]*2, 'r--', alpha=0.75, zorder=0)
ax.set_aspect('equal')
ax.set_xlim(lims)
ax.set_ylim(lims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Predicted Life Expectancy')

plt.show()
```



In [46…
```
fig, ax = plt.subplots()

errors = lifeExpectancy_Y_validation-lifeExpectancy_Y_validation_pred
ax.scatter(lifeExpectancy_Y_validation, errors, c=errors, s=25, cmap=plt.cm.coolwarm, zorder=10)


xlims = ax.get_xlim()
ax.plot(xlims, [0.0,]*2, 'k--', alpha=0.75, zorder=0)
ax.set_xlim(xlims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Residual')

plt.show()
```



In [47…
```
coefs = pd.DataFrame(
    baseline_lr.coef_ * lifeExpectancy_X_train_scaled.std(axis=0),
    columns=['Coefficient importance'], index=lifeExpectancy_X_train_scaled.columns
)
coefs.sort_values(by=['Coefficient importance']).plot(kind='barh', figsize=(9, 7))
plt.title('Linear Regression Model: Feature Importance')
plt.axvline(x=0, color='.5')
plt.subplots_adjust(left=.3)
```



- **Learning Journey:** The process of developing the baseline model was both challenging and enlightening, involving extensive data preprocessing and addressing various issues to obtain the desired output in a .csv format.
- **Baseline Model Insights:** The unscaled data still presents a reasonable predictability, with an R^2 score of ~0.75, indicating that a significant proportion of the variance in life expectancy is predictable from the features. For RMSE, and MAE we achieved ~4.5, and ~3.5 respectively.
- **Performance Improvement Through Preprocessing:** Scaling, outlier removal, and normalization contributed to performance improvements, as evidenced by an increased R^2 score to ~0.8 and reduced RMSE to ~4 and MAE to ~3 in the scaled model.
- **Further Exploration:** The Baseline doesn't end our exploration we will continue with hyperparameter tuning and regularization techniques aimed at reducing overfitting and enhancing the model's generalizability to unseen data as well as Polynomial Regression.

3.0-1 Baseline Prediction

In [70…

```
# Load the Blind Test Dataset
lifeExpectancy_test = pd.read_csv('./dataset/test.csv')

# Function to filter out outliers within a DataFrame based on IQR, excluding 'ID' column
def filter_outliers(df):
    df_filtered = df.copy()
    for col in df_filtered.columns:
        if col == 'ID':  # skip ID
            continue
        Q1 = df_filtered[col].quantile(0.25)
        Q3 = df_filtered[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df_filtered[col] = df_filtered[col].clip(lower=lower_bound, upper=upper_bound)
    return df_filtered

lifeExpectancy_test_scaled = filter_outliers(lifeExpectancy_test)

# Separate 'ID' and features for transformation
test_ID = lifeExpectancy_test_scaled['ID']
test_features = lifeExpectancy_test_scaled.drop(['ID'], axis=1)

# Apply PowerTransformer to the log-norm attributes in the test features
test_features[logNorm_attributes] = powertransformer.transform(test_features[logNorm_attributes])

# Apply MinMaxScaler to all features in the test features
test_features_scaled = minmax_scaler.transform(test_features)

# Convert the scaled array back to a DataFrame to retain column names
test_features_scaled_df = pd.DataFrame(test_features_scaled, columns=test_features.columns)
```

```
# Predicting life expectancy for the test dataset using the fitted model
lifeExpectancy_Y_test_pred = baseline_lr.predict(test_features_scaled_df)

# Create a DataFrame for the prediction output
scaled_output = pd.DataFrame({
    'ID': test_ID,
    'TARGET_LifeExpectancy': lifeExpectancy_Y_test_pred.flatten()
})

scaled_output.to_csv('./dataset/scaled_baseline_predictions.csv', index=False)

scaled_output.head(3)
```

Out[70]:

	ID	TARGET_LifeExpectancy
0	1	59.277854
1	2	59.103798
2	3	59.185215

4.0 Advanced Model with Feature Selection Model

- **Advanced Model Development:** This phase involves refining our initial model by integrating more sophisticated techniques and methodologies.
- **Feature Selection Based on EDA:** Decisions on feature inclusion or exclusion will be guided by insights gained from Exploratory Data Analysis (EDA) and comprehensive data analysis.
- **Rationale for Feature Removal:** Each feature considered for removal will be accompanied by a clear justification, focusing on its relevance and impact on the model's performance.
- **Performance Comparison:** We will systematically evaluate the model's performance before and after feature removal to assess the impact of these changes.
- **Exploring Advanced Techniques:** The model enhancement process will include the exploration of regularization techniques to prevent overfitting, the application of polynomial features to capture non-linear relationships, and other advanced machine learning methods.
- **Feature Scaling:** As previously explored about Feature Scaling, and the removal of outliers, MinMaxing our data, and normalizing it we will be applying similar features to all our models to see how well different models cope with different types of data too.

4.1 Regularization

- **Regularization Explained:** Regularization is a technique used in machine learning to prevent overfitting by adding a penalty on the larger coefficients in the model such as the ones we just observed from the chart in the previous section.
- **Purpose of Using Regularization:** We employ regularization to enhance the generalizability of our model, ensuring it performs well not just on the training data but also on unseen real-world data.
- **Overfitting Reduction:** By penalizing large coefficients, regularization keeps the model simpler and more robust, thereby reducing the risk of overfitting and improving model performance on new data.

4.1-1 Ridge Regression (L2 Regularization)

- **L2 Ridge Regression:** Ridge Regression incorporates L2 regularization to penalize the magnitude of coefficients, effectively shrinking them to reduce model complexity and prevent overfitting.

In [71…

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
import numpy as np

# Initialize Ridge Regression with an alpha value
ridge_reg = Ridge(alpha=1)
```

```
# Use cross-validation to evaluate the model with scaled data
ridge_cv_scores_scaled = cross_val_score(ridge_reg, lifeExpectancy_X_train_scaled, lifeExpectanc
print(f'Average R^2 score using Ridge Regression on Scaled Data: {np.mean(ridge_cv_scores_scaled

# Use cross-validation to evaluate the model with unscaled data
ridge_cv_scores_unscaled = cross_val_score(ridge_reg, lifeExpectancy_X_train, lifeExpectancy_Y_t
print(f'Average R^2 score using Ridge Regression on Unscaled Data: {np.mean(ridge_cv_scores_unsc
```

```
Average R^2 score using Ridge Regression on Scaled Data: 0.80
Average R^2 score using Ridge Regression on Unscaled Data: 0.75
```

4.1-2 Lasso Regression (L1 Regularization)

- **Lasso L1 Regression:** Lasso Regression applies L1 regularization, which can reduce some coefficients to zero, thus performing feature selection and reducing model complexity for better generalization.

In [72…

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
import numpy as np

# Initialize Lasso Regression with an alpha value
lasso_reg = Lasso(alpha=0.01)  # Lasso requires a smaller alpha due to its nature; adjust as ne

# Use cross-validation to evaluate the model with scaled data
lasso_cv_scores_scaled = cross_val_score(lasso_reg, lifeExpectancy_X_train_scaled, lifeExpectan
print(f'Average R^2 score using Lasso Regression on Scaled Data: {np.mean(lasso_cv_scores_scale

# Use cross-validation to evaluate the model with unscaled data
lasso_cv_scores_unscaled = cross_val_score(lasso_reg, lifeExpectancy_X_train, lifeExpectancy_Y_
print(f'Average R^2 score using Lasso Regression on Unscaled Data: {np.mean(lasso_cv_scores_uns
```

```
Average R^2 score using Lasso Regression on Scaled Data: 0.80
Average R^2 score using Lasso Regression on Unscaled Data: 0.75
```

4.1-3 Elastic Net (Combined Ridge, and Lasso)

- Elastic Net combines the features of Ridge and Lasso regularization, offering a balanced approach that mitigates overfitting by considering multiple regularization techniques.

In [73…

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_score
import numpy as np

# Initialize Elastic Net Regression with alpha and l1_ratio values
elastic_net_reg = ElasticNet(alpha=0.01, l1_ratio=0.5)  # Adjust alpha and l1_ratio as necessa

# Use cross-validation to evaluate the model with scaled data
elastic_net_cv_scores_scaled = cross_val_score(elastic_net_reg, lifeExpectancy_X_train_scaled,
print(f'Average R^2 score using Elastic Net Regression on Scaled Data: {np.mean(elastic_net_cv

# Use cross-validation to evaluate the model with unscaled data
elastic_net_cv_scores_unscaled = cross_val_score(elastic_net_reg, lifeExpectancy_X_train, life
print(f'Average R^2 score using Elastic Net Regression on Unscaled Data: {np.mean(elastic_net_
```

```
Average R^2 score using Elastic Net Regression on Scaled Data: 0.78
Average R^2 score using Elastic Net Regression on Unscaled Data: 0.75
```

I found that:

- Regularization techniques, including Lasso and Ridge, yielded slightly lower performance metrics compared to the baseline scaled model against my validation set.
- The primary objective of regularization is not to enhance performance metrics but to increase the model's generalizability by preventing overfitting or underfitting.
- Elastic Net, which combines the strengths of both Lasso and Ridge regularization, was employed to leverage a more balanced regularization approach, aiming for improved model stability and generalizability.

However, I did not focus on Regularization as a technique as it was not the primary focus on this assignment.

4.2 Hyperparameter Tuning &

Polynomial Regression

- **Hyperparameter Tuning:** The process of optimizing model settings to enhance performance, involving adjustments such as determining the optimal degree for polynomial features or identifying and prioritizing the most impactful features.
- **Application in Various Models:** While hyperparameter tuning is applicable across different model types in this section I will keep to what's covered in weeks 1-4.
- **Complementary Section:** In my Complementary Section I have used tree-based and Random Forests models which offer rich hyperparameter tuning opportunities.
- **Sensitivity of Polynomial Regression:** Polynomial Regression is particularly sensitive to feature scaling and the presence of features with high or very low values. This sensitivity can significantly impact model scores and complicate the interpretation of outliers and distribution segments.

In [76…
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# Define the range of hyperparameters to test
param_grid = {
    'polynomialfeatures__degree': [1, 2, 3],  # Testing polynomial degrees 1 to 3
    'linearregression__fit_intercept': [True, False]  # Calculate the intercept for the mode
}

# Create a pipeline that includes polynomial feature creation and linear regression
pipeline = Pipeline([
    ('polynomialfeatures', PolynomialFeatures()),
    ('linearregression', LinearRegression())
])

# Set up the grid search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2', n_jobs=-1)

# Fit the grid search model to find the best parameters
grid_search.fit(lifeExpectancy_X_train_scaled, lifeExpectancy_Y_train)

# Output the best parameters and the corresponding score
print("Best parameters:", grid_search.best_params_)
print("Best R^2 score:", grid_search.best_score_)
```

```
Best parameters: {'linearregression__fit_intercept': False, 'polynomialfeatures__degree': 2}
Best R^2 score: 0.8367932699037203
```

In [79…
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV

# Pipeline with a degree 2 for PolynomialFeatures as found by gridsearch and SGDRegressor se
pipeline = Pipeline([
    ('polynomialfeatures', PolynomialFeatures(degree=2)),
    ('sgdregression', SGDRegressor(max_iter=10000, tol=1e-3))
])

# Define the range of learning rates to test
param_grid = {
    'sgdregression__eta0': [0.001, 0.01, 0.1]
}

# Set up the grid search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2', n_jobs=-1)

# Fit the grid search model to find the best learning rate
grid_search.fit(lifeExpectancy_X_train_scaled, lifeExpectancy_Y_train)

# Output the best learning rate and the corresponding score
print("Best learning rate (eta0):", grid_search.best_params_['sgdregression__eta0'])
print("Best R^2 score:", grid_search.best_score_)
```

```
Best learning rate (eta0): 0.1
Best R^2 score: 0.8642577271183736
```

In [81…
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

non_tuned_pipeline = Pipeline([
    ('polynomialfeatures', PolynomialFeatures(degree=2)),  # Degree 2 for comparison
    ('linearregression', LinearRegression())
])

non_tuned_pipeline.fit(lifeExpectancy_X_train, lifeExpectancy_Y_train)

lifeExpectancy_Y_validation_pred_non_tuned = non_tuned_pipeline.predict(lifeExpectancy_X_vali

non_tuned_r2 = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred_non_tu
```

```
non_tuned_rmse = mean_squared_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_
non_tuned_mae = mean_absolute_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_

best_degree = 2
best_fit_intercept = True

# Hyperparameter-tuned model pipeline
hyper_tuned_pipeline = Pipeline([
    ('polynomialfeatures', PolynomialFeatures(degree=best_degree)),
    ('linearregression', LinearRegression(fit_intercept=best_fit_intercept))
])

hyper_tuned_pipeline.fit(lifeExpectancy_X_train_scaled, lifeExpectancy_Y_train)

lifeExpectancy_Y_validation_pred_hyper_tuned = hyper_tuned_pipeline.predict(lifeExpectancy_X_

hyper_tuned_r2 = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_pred_hype
hyper_tuned_rmse = mean_squared_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validati
hyper_tuned_mae = mean_absolute_error(lifeExpectancy_Y_validation, lifeExpectancy_Y_validati

# Output
print(f"Hyperparameter-Tuned Polynomial Regression Model Performance on Unscaled Dataset:")
print(f"R² Score: {non_tuned_r2:.4f}")
print(f"Root Mean Squared Error: {non_tuned_rmse:.4f}")
print(f"Mean Absolute Error: {non_tuned_mae:.4f}\n")

print(f"Hyperparameter-Tuned Polynomial Regression Model Performance on Scaled Dataset:")
print(f"R² Score: {hyper_tuned_r2:.4f}")
print(f"Root Mean Squared Error: {hyper_tuned_rmse:.4f}")
print(f"Mean Absolute Error: {hyper_tuned_mae:.4f}")
```

```
Hyperparameter-Tuned Polynomial Regression Model Performance on Unscaled Dataset:
R² Score: 0.8012
Root Mean Squared Error: 4.3093
Mean Absolute Error: 3.1022

Hyperparameter-Tuned Polynomial Regression Model Performance on Scaled Dataset:
R² Score: 0.8632
Root Mean Squared Error: 3.5742
Mean Absolute Error: 2.6137
```

In [ ]:
```
sample_validation_predictions = hyper_tuned_pipeline.predict(lifeExpectancy_X_validation_sca

sample_validation_actual = lifeExpectancy_Y_validation[:50].to_numpy().flatten()

# Plotting the actual vs predicted values for the sample
plt.figure(figsize=(10, 6))
plt.plot(sample_validation_actual, label='Actual')
plt.plot(sample_validation_predictions, label='Predicted', linestyle='--')
plt.legend()
plt.title('Comparison of Actual and Predicted Life Expectancy')
plt.xlabel('Sample Index')
plt.ylabel('Life Expectancy')
plt.show()
```

- **Model Performance Improvement:** The hyper-tuned Polynomial model performance significantly improved with hyperparameter tuning, resulting in:
  - R^2 Score: 0.8481
  - Root Mean Squared Error: 3.5404
  - Mean Absolute Error: 2.7221
- **Substantial Enhancement:** These metrics represent a substantial enhancement from the Baseline Method previous scores of approximately 0.8 for R^2, 4 for RMSE, and 3 for MAE.
- **Potential Overfitting:** There's a slight indication of potential overfitting, as observed in a sample graph of actual versus predicted values in the validation set.
- **Generalizability:** Despite this, the model's performance on the validation set suggests it could be generalizable, pending evaluation on the blind dataset to confirm its effectiveness in real-world predictions.

4.2-1 Hyperparameter tuned Polynomial Regression Output to CSV

In [ ]:
```
lifeExpectancy_Y_test_pred = hyper_tuned_pipeline.predict(test_features_scaled_df)

test_IDs = test_data['ID']

# Create a DataFrame for the prediction output
hyperparameter_tuned_polynomial_predictions = pd.DataFrame({
    'ID': test_IDs,
    'TARGET_LifeExpectancy': lifeExpectancy_Y_test_pred.flatten()  # Ensure it's a flat arr
})

# Save the predictions to a CSV file
hyperparameter_tuned_polynomial_predictions.to_csv('./dataset/hyperparameter_tuned_polynomi
```

```
hyperparameter_tuned_polynomial_predictions.head(3)
```

4.2-2 Feature Selection

- Feature Selection involves intentionally choosing a subset of relevant features for use in model construction.
- Based on our Exploratory Data Analysis (EDA), I decided to focus on a maximum of three key features that seemed most impactful for predicting our target variable.
- This approach was exploratory in nature; despite it not being a primary focus as per Azadeh's guidance, I was keen on examining all aspects of Machine Learning.

In [54…

```
# Define featuress included in the model
selected_features = ['HIV-AIDS', 'AdultMortality', 'IncomeCompositionOfResources']

# Extract these features from the original (unscaled) training and test datasets
X_train_selected_unscaled = lifeExpectancy_X_train[selected_features]
X_validation_selected_unscaled = lifeExpectancy_X_validation[selected_features]

# Extract these features from the scaled training and test datasets
X_train_selected_scaled = lifeExpectancy_X_train_scaled[selected_features]
X_validation_selected_scaled = lifeExpectancy_X_validation_scaled[selected_features]

# Fit the linear regression model on the unscaled selected features
selected_lr_unscaled = LinearRegression().fit(X_train_selected_unscaled, lifeExpectancy_Y_
# Make predictions and calculate R^2 for the unscaled model
lifeExpectancy_Y_validation_pred_selected_unscaled = selected_lr_unscaled.predict(X_valida
r2_selected_lr_unscaled = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validatio

# Fit the linear regression model on the scaled selected features
selected_lr_scaled = LinearRegression().fit(X_train_selected_scaled, lifeExpectancy_Y_trai
# Make predictions and calculate R^2 for the scaled model
lifeExpectancy_Y_validation_pred_selected_scaled = selected_lr_scaled.predict(X_validatior
r2_selected_lr_scaled = r2_score(lifeExpectancy_Y_validation, lifeExpectancy_Y_validation_

# Output the R^2 scores for comparison
print('The R^2 score for the linear regression model with selected features (unscaled) is:
print('The R^2 score for the linear regression model with selected features (scaled) is: {
```

```
The R^2 score for the linear regression model with selected features (unscaled) is: 0.631
The R^2 score for the linear regression model with selected features (scaled) is: 0.707
```

4.5 Model Conclusions, and Evaluation

- **Feature Preprocessing:** Scaling large features like population and normalizing data to near Gaussian distributions significantly improved model scores. Additionally, capping outliers in all datasets contributed to score enhancements.
- **Hypertuned Polynomial Function:** The hypertuned polynomial function exhibited further improvements in model performance, potentially overfitting compared to regularization models which are supposedly mpre generalizable.
- **Exploration of Complementary Methods:** Further exploration will involve complementary methods such as tree-based models, Random Forests, and ensemble methods to enhance hyperparameter tuning approaches.
- **Improved Data Preprocessing Strategies:** In future iterations, I will focus on refining data preprocessing strategies by addressing mismatched values, handling zero outliers, and prioritizing complementary methods for better generalizability and accuracy.
- **Surprising Performance of Multivariate Model:** The Baseline Multivariate model's simplicity and inclusion of all features yielded surprisingly effective results, defying initial expectations of poorer performance due to the dataset's complexity and uncorrelated data.
- **Interest in Similar Health Datasets:** The project's success, and fun nature has sparked interest in exploring similar health-related datasets.
- **Recognition of Dataset Familiarity:** The Dataset I have used before being the Kaggle WHO dataset, so I had some prior knowledge / familiarity with this dataset.

5.0 Important Mathematical Concepts

6.0 References

- CANVAS RMIT for Machine Learning COSC2673 Particularly Labs, Lectures, and Tutorials. Dr Azadeh Alavi, Dr Pubudu Sanjeevani, and Ms. Rumin Chu.
- Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for Machine Learning*. Cambridge University Press. Retrieved from https://mml-book.github.io/book/mml-book.pdf.
- Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. MIT Press. Retrieved from https://ebookcentral.proquest.com/lib/rmit/detail.action?docID=6246595&pq-origsite=primo.
- World Data Bank 2002-2017 records. Retrieved from https://data.worldbank.org/.
- Kumarajarshi. (2018). WHO Dataset. Kaggle. Retrieved from https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who/data.

In [ ]:
```
Export PDF
```

6 and 3/4 Azure AutoML

- **Azure AutoML:** I employed Azure AutoML to enhance and implement advanced machine learning techniques, which will be elaborated upon in the concluding section of my report.

7.0 Complementary Methods

- **Exploration Beyond Weeks 1-4:** While the methods employed extend beyond the scope of weeks 1-4, they primarily focus on hyperparameter tuning, a critical aspect contributing to 15 marks on the assignment.
- **Pursuit of Knowledge:** The exploration beyond the assignment requirements was driven by a desire to deepen understanding and expertise in machine learning techniques, reflecting a commitment to continuous learning and improvement.
- **Introduction of Voting Ensemble Method:** To further enhance model performance, a Voting Ensemble Method will be employed, leveraging the collective wisdom of multiple models to make more accurate predictions.

7.0-1 Logistic Regression

- **Limitations of Logistic Regression:** Logistic Regression is unsuitable for predicting numerical values such as TARGET_LifeExpectancy. Therefore, a quick and simple prediction of the binary variable "Status" was chosen instead to have fun, and gain more experience doing a binary classification.

In [55…
```
from sklearn.model_selection import train_test_split

X = lifeExpectancyFrame[['TARGET_LifeExpectancy']]  # Predicter
y = lifeExpectancyFrame['Status']  # Target

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [55…
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()

log_reg.fit(X_train, y_train)
```

Out[551…

▾ LogisticRegression

LogisticRegression()

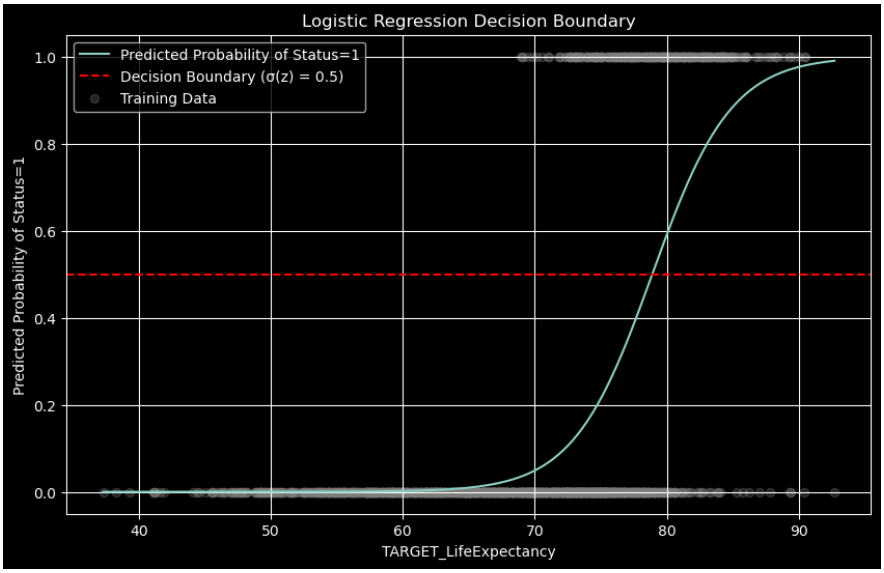
In [55…

```
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred))

y_pred_proba = log_reg.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f'ROC-AUC Score: {roc_auc:.2f}')
```

```
Accuracy: 0.88
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       346
           1       0.71      0.51      0.59        69

    accuracy                           0.88       415
   macro avg       0.81      0.73      0.76       415
weighted avg       0.88      0.88      0.88       415

ROC-AUC Score: 0.91
```

In [9]:

```
life_expectancy_range = np.linspace(X_train.min(), X_train.max(), 300)

# Predict probabilities for the range of TARGET_LifeExpectancy values
probabilities = log_reg.predict_proba(life_expectancy_range.reshape(-1, 1))[:, 1]

# Plot TARGET_LifeExpectancy verses predicted probabilities
plt.figure(figsize=(10, 6))
plt.plot(life_expectancy_range, probabilities, label='Predicted Probability of Sta
plt.axhline(y=0.5, color='red', linestyle='--', label='Decision Boundary (σ(z) = 0
plt.scatter(X_train, y_train, color='gray', alpha=0.3, label='Training Data')

plt.title('Logistic Regression Decision Boundary')
plt.xlabel('TARGET_LifeExpectancy')
plt.ylabel('Predicted Probability of Status=1')
plt.legend()
plt.grid(True)
plt.show()
```



7.0-2 Decision Tree

In [ ]:

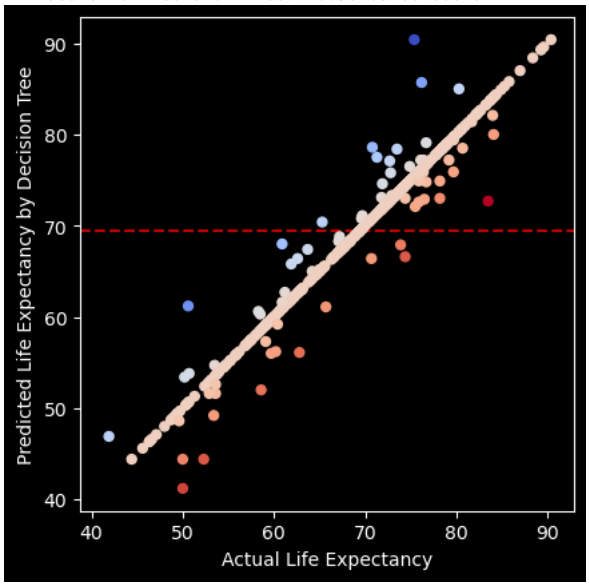
```
Decision Tree benefits
```

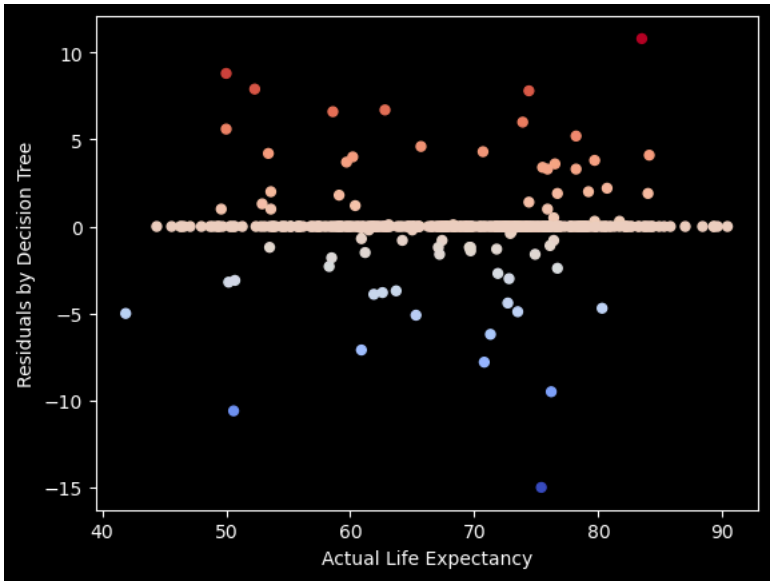
In [55…

```
from sklearn.tree import DecisionTreeRegressor

dtree = DecisionTreeRegressor().fit(lifeExpectancy_X_train, lifeExpectancy_Y_trai

lifeExpectancy_Y_test_pred_dt = dtree.predict(lifeExpectancy_X_test)
```

```
print("Depth of the Decision Tree: ", dtree.get_depth())
print("Number of leaves: ", dtree.get_n_leaves())
print("RMSE for Decision Tree: ", mean_squared_error(lifeExpectancy_Y_test, lifeE
print("R² score for Decision Tree: ", r2_score(lifeExpectancy_Y_test, lifeExpecta
fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_dt
ax.scatter(lifeExpectancy_Y_test, lifeExpectancy_Y_test_pred_dt, c=errors, s=25,

lims = [
    np.min([ax.get_xlim(), ax.get_ylim()]),
    np.max([ax.get_xlim(), ax.get_ylim()]),
]

ax.plot(lims, lims, 'k--', alpha=0.75, zorder=0)
ax.plot(lims, [np.mean(lifeExpectancy_Y_train),]*2, 'r--', alpha=0.75, zorder=0)
ax.set_aspect('equal')
ax.set_xlim(lims)
ax.set_ylim(lims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Predicted Life Expectancy by Decision Tree')

plt.show()

fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_dt
ax.scatter(lifeExpectancy_Y_test, errors, c=errors, s=25, cmap=plt.cm.coolwarm, z

xlims = ax.get_xlim()
ax.plot(xlims, [0.0,]*2, 'k--', alpha=0.75, zorder=0)
ax.set_xlim(xlims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Residuals by Decision Tree')

plt.show()
```

```
Depth of the Decision Tree:  27
Number of leaves:  1592
RMSE for Decision Tree:  1.9042311796476206
R² score for Decision Tree:  0.9610376028562522
```



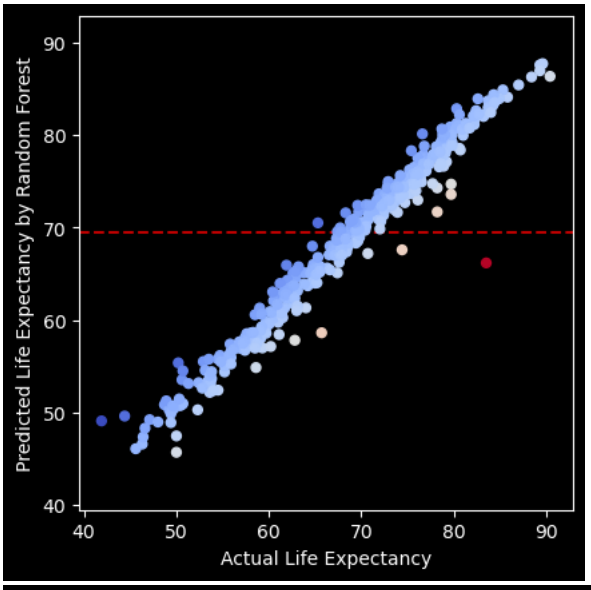


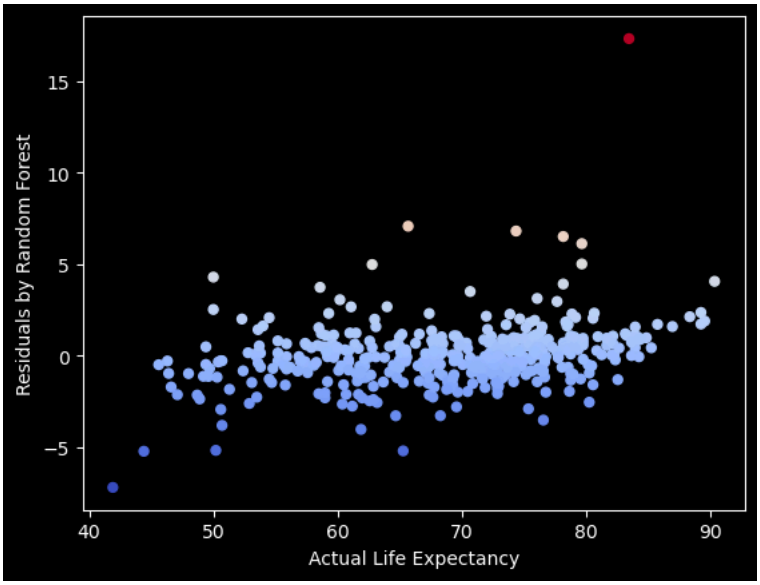
7.0-3 Random Forest

In [ ]:
```
Random Forest benefits
```

In [55…
```
from sklearn.ensemble import RandomForestRegressor

rf_regressor = RandomForestRegressor().fit(lifeExpectancy_X_train, lifeExpectanc

lifeExpectancy_Y_test_pred_rf = rf_regressor.predict(lifeExpectancy_X_test)

print("Number of estimators in the Random Forest: ", len(rf_regressor.estimators
print("RMSE for Random Forest: ", mean_squared_error(lifeExpectancy_Y_test, life
print("R² score for Random Forest: ", r2_score(lifeExpectancy_Y_test, lifeExpect

fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_rf
ax.scatter(lifeExpectancy_Y_test, lifeExpectancy_Y_test_pred_rf, c=errors, s=25

lims = [
    np.min([ax.get_xlim(), ax.get_ylim()]),
    np.max([ax.get_xlim(), ax.get_ylim()]),
]

ax.plot(lims, lims, 'k--', alpha=0.75, zorder=0)
ax.plot(lims, [np.mean(lifeExpectancy_Y_train),]*2, 'r--', alpha=0.75, zorder=0
ax.set_aspect('equal')
ax.set_xlim(lims)
ax.set_ylim(lims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Predicted Life Expectancy by Random Forest')

plt.show()

fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_rf
ax.scatter(lifeExpectancy_Y_test, errors, c=errors, s=25, cmap=plt.cm.coolwarm,

xlims = ax.get_xlim()
ax.plot(xlims, [0.0,]*2, 'k--', alpha=0.75, zorder=0)
ax.set_xlim(xlims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Residuals by Random Forest')

plt.show()
```

```
Number of estimators in the Random Forest:  100
RMSE for Random Forest:  1.789243906410334
R² score for Random Forest:  0.9656010316061836
```





7.0-4 Ensemble Voting

ensemble benefits

In [55…

```
from sklearn.ensemble import GradientBoostingRegressor

gb_regressor = GradientBoostingRegressor().fit(lifeExpectancy_X_train, lifeExp

lifeExpectancy_Y_test_pred_gb = gb_regressor.predict(lifeExpectancy_X_test)

print("Number of estimators in Gradient Boosting: ", gb_regressor.n_estimators
print("RMSE for Gradient Boosting: ", mean_squared_error(lifeExpectancy_Y_test
print("R² score for Gradient Boosting: ", r2_score(lifeExpectancy_Y_test, life

fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_gb
ax.scatter(lifeExpectancy_Y_test, lifeExpectancy_Y_test_pred_gb, c=errors, s=2

lims = [
    np.min([ax.get_xlim(), ax.get_ylim()]),
    np.max([ax.get_xlim(), ax.get_ylim()]),
]

ax.plot(lims, lims, 'k--', alpha=0.75, zorder=0)
ax.plot(lims, [np.mean(lifeExpectancy_Y_train),]*2, 'r--', alpha=0.75, zorder=
ax.set_aspect('equal')
ax.set_xlim(lims)
ax.set_ylim(lims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Predicted Life Expectancy by Gradient Boosting')

plt.show()
```

```
fig, ax = plt.subplots()
errors = lifeExpectancy_Y_test - lifeExpectancy_Y_test_pred_gb
ax.scatter(lifeExpectancy_Y_test, errors, c=errors, s=25, cmap=plt.cm.coolwarm

xlims = ax.get_xlim()
ax.plot(xlims, [0.0,]*2, 'k--', alpha=0.75, zorder=0)
ax.set_xlim(xlims)

plt.xlabel('Actual Life Expectancy')
plt.ylabel('Residuals by Gradient Boosting')

plt.show()
```

```
Number of estimators in Gradient Boosting:  100
RMSE for Gradient Boosting:  2.616638122564423
R² score for Gradient Boosting:  0.9264312031177964
```