MACHINE LEARNING (COSC2673)

ASSIGNMENT 1

AHNAF TAUSIF
S3890097

Date: 10/04/24

Exploratory Data Analysis (EDA):

An exploratory data analysis (EDA) was conducted on the train.csv file. Here are the key findings:

- The describe () function was utilized to view a summary of the statistics of the dataset (shown below Fig 1.0) such as the count, mean, median and mode values. There were no discrepancies in the data as the count for all the features were 2071.

```
data.describe()
```

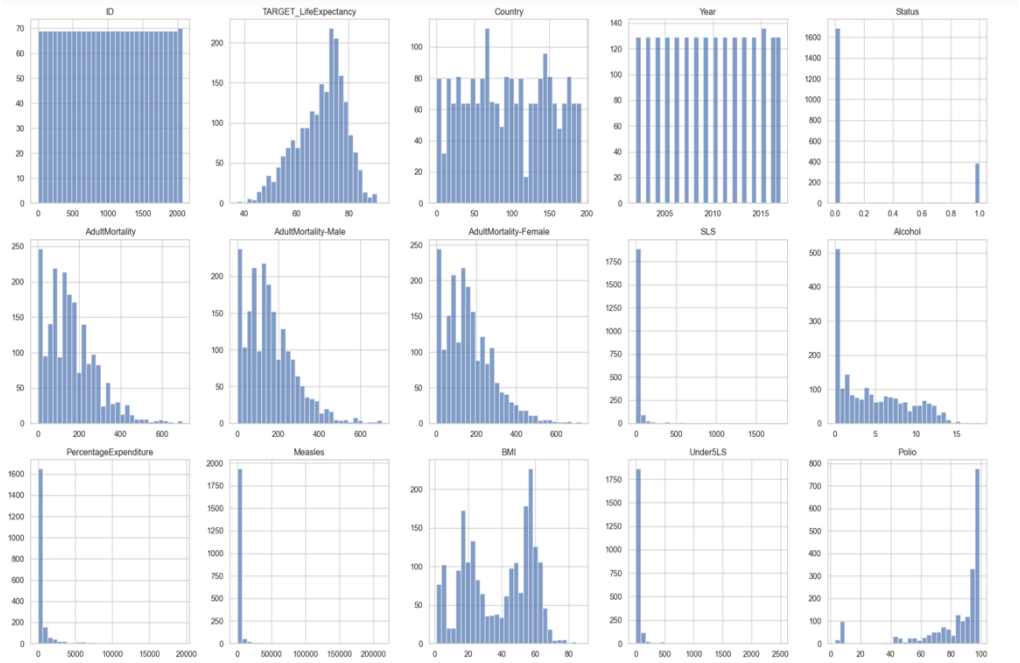
	ID	TARGET_LifeExpectancy	Country	Year	Status	AdultMortality	AdultMortality-Male	AdultMortality-Female	SLS	Alcohol
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000
mean	1036.000000	69.274505	95.360212	2009.518590	0.185418	162.833897	161.908257	163.759536	33.079672	4.696379
std	597.990524	9.482281	54.861641	4.614147	0.388730	118.872170	119.442235	118.800292	135.832868	4.205888
min	1.000000	37.300000	0.000000	2002.000000	0.000000	1.000000	0.000000	2.000000	0.000000	0.010000
25%	518.500000	63.000000	50.000000	2006.000000	0.000000	74.000000	74.000000	74.000000	0.000000	0.615000
50%	1036.000000	71.200000	94.000000	2010.000000	0.000000	144.000000	142.000000	144.000000	3.000000	3.830000
75%	1553.500000	76.000000	144.000000	2014.000000	0.000000	228.000000	228.000000	230.000000	22.000000	7.840000
max	2071.000000	92.700000	192.000000	2017.000000	1.000000	699.000000	704.000000	722.000000	1800.000000	17.870000

- Checked to see if any null values were present in the dataset using the info() function(shown below Fig 1.1). We observed that all the features had a non-null value and as a result we did not have to do any changes to the contents of the dataset.

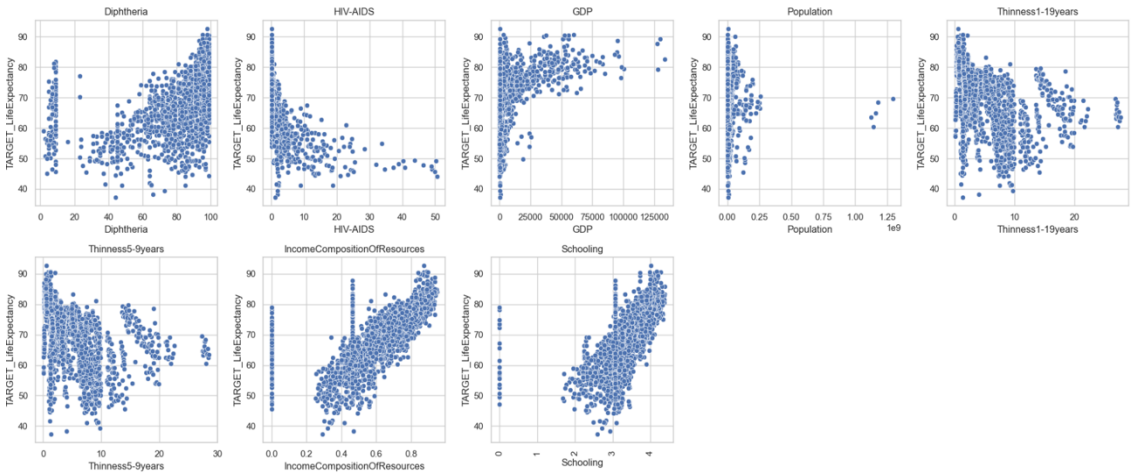
```
print(data.info())
```

```
<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
None
```

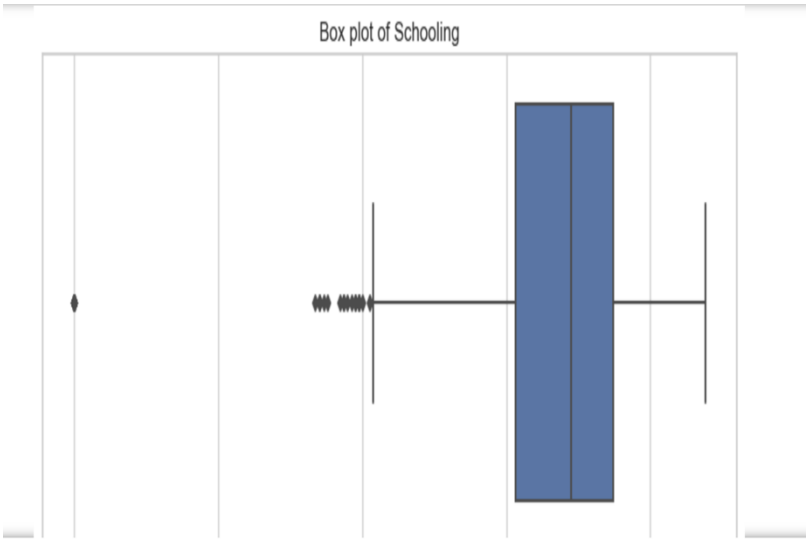
- **Histograms**:
  - When histograms were plotted for all the features (shown below Fig 1.2), it was observed that most of the histograms were skewed. Features such as AdultMortality, alcohol, PercentageExpenditure, TotalExpenditure, are right skewed. And the rest of the features, excluding the Target_LifeExpectancy, country, year and status were all left skewed. This observation indicated that the scaling of the graph needed to be adjusted.
  - Most of the features have different scales with some so small that no conclusive observation could be made.
  - We can clearly identify status is a categorical value.
  - Features have linear relationship with each other.



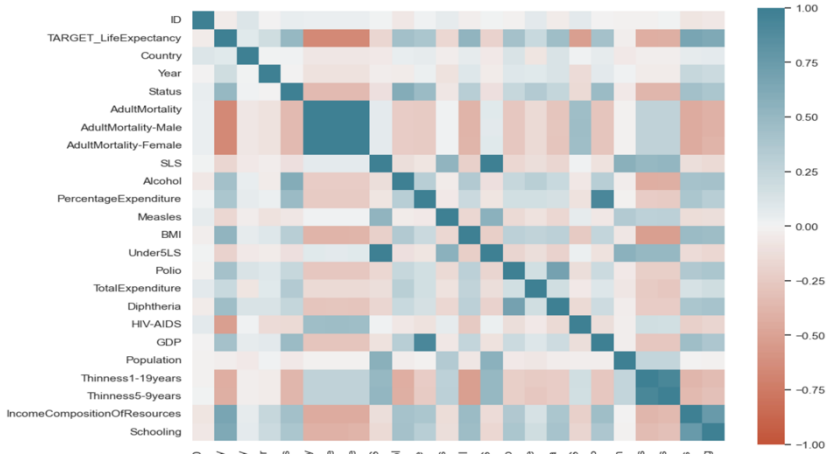
- **Scatter plots**:
  - After plotting the scatter graph (some examples shown below Fig 1.3) of all the feature against Target_LifeExpectancy, I observed that the graphs showed similar pattern as the histograms. The left skewed features showed a positive correlation whereas the right skewed graphs showed negative correlation. Some of these can be expected, for example, AdultMortality has a negative correlation with Target_LifeExpectancy as a higher adult mortality rate would indicate the life expectancy is short for that country.



- **Boxplot**:
  - To identify any outliers in the features, a box plot graph was created (example shown below Fig 1.4).
  - Many features such as AdultMortality, TotalExpenditure and Schooling had outliers present which highlights that these outliers need to be processed to create a model which could predict the life expectancy accurately.



- **Correlation Heat map**:
  - A correlation heat map was constructed (shown below Fig 1.5) to observe how much one feature has an influence on another feature.



  - To make it easier, a code was implemented (shown below) to directly find the features that have a correlation of more than or equal to 0.8.

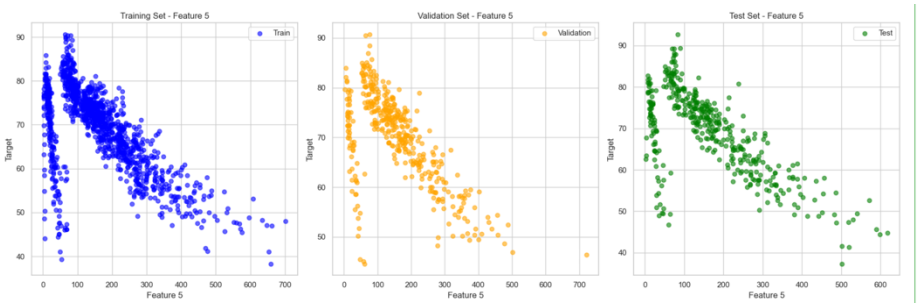
```
corr_matrix = data.corr()

# Find features with correlation greater than 0.8
high_corr = (corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
                        .stack()
                        .sort_values(ascending=False))

# Filter pairs with high correlation
high_corr_pairs = high_corr[high_corr >= 0.8]

print("Highly correlated feature pairs (correlation coefficient >= 0.8):")
print(high_corr_pairs)
```

  - From the heatmap, we observed that some of the features, such as SLS and Under5LS has high correlation.
  - However, these high correlations were all expected as the features mostly are medical (e.g. BMI) and economic (e.g. GDP) traits which go hand in hand in the real world. From these observations we understand that to produce accurate predictions, we need to perform regularization to manage this issue of multicollinearity.

Train, Validation and Test split:

- The dataset was split into training, validation, and testing sets in the ratio 60% training, 20% to validation and 20% to testing in which the train set has 1242 instances, validation set has 414 instances and test has 415 instances.
    - I chose the random state to be my student ID (3890097) to make the split unique for me.
    - Below is the code with which I achieved this:

```
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=3890097
)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=3890097
)
```



- From the scatter plot above, we can observe that the 3 splits made have similar patterns and spread. Therefore, we can eliminate the possibility of the splits having any bias preexisting in the data.

Baseline model:

- I have decided to use Linear regression to be my base model for the following reasons:
    - From our histograms and scatter plots, we observed that the features have linear relationships with each other.
    - Introduces a penalty to reduce the impact of highly correlated features.
    - Linear regression models are simple to implement.
    - They are easy to interpret as it provides coefficients that indicate the relationship between predictors and the target variable.
    - Linear regression models require few assumptions thus making it ideal to use to set a benchmark to compare other models to.

- The two figures on the right present the code used to train the baseline model and its outcome. Let us look at what we can determine from these results:
  We calculated the Mean Squared Error (MSE) for the test set is 23.80947 while the MSE for the validation set is 18.49646708 which indicates slight possible overfitting for the validation set or it could be a result of natural variance in the distribution of data in the test and validation set.
  We calculated the R-squared value for the test set and validation set were very close to one another. From this we can understand that the model can consistently explain the variance across the datasets.

```python
# Initialize the model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the validation data
val_predictions = model.predict(X_val)

# Evaluate the model on the validation set
val_mse = mean_squared_error(y_val, val_predictions)
val_r_squared = r2_score(y_val, val_predictions)
print(f"Validation Mean Squared Error: {val_mse}")
print(f"Validation R-squared value: {val_r_squared}")

# Check if performance criteria are met
if val_mse < 25 and val_r_squared > 0.7:

    print("\nCriteria met. Proceeding to evaluate on the test set.\n")

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

    # Evaluate the model on the test set
    test_mse = mean_squared_error(y_test, test_predictions)
    test_r_squared = r2_score(y_test, test_predictions)
    print(f"Test Mean Squared Error: {test_mse}")
    print(f"Test R-squared value: {test_r_squared}")

    # Calculate residuals based on the test set
    residuals = y_test - test_predictions

    # Calculate standard deviation of residuals
    std_residuals = np.std(residuals)
    print(f"Standard Deviation of Residuals: {std_residuals}")

else:
    print("Criteria not met. Consider revising the model or parameters.")
```

```
Validation Mean Squared Error: 18.496467086264893
Validation R-squared value: 0.7748100997336658

Criteria met. Proceeding to evaluate on the test set.

Test Mean Squared Error: 23.80947775379669
Test R-squared value: 0.7662577793419321
Standard Deviation of Residuals: 4.878795414461385
```

Feature Scaling:

Through our EDA, it was concluded that the features in the dataset were skewed and had varying scales. This is an issue as it creates bias in the dataset. To tackle this issue and normalize the data, I used Power Transformer followed by min-max scaling.

- Power transformer helps reduce the variance and makes the data more Gaussian-like.
- Min-max scaling allows us to rescale the features to a fixed range. Both combined allows us to train our model to provide a more accurate prediction.
- We will apply the Power Transformer followed by min-max scaling through the following code:

```python
# Apply Power Transformation to the skewed features
X_train_power_transformed = power_transformer.fit_transform(X_train)
X_val_power_transformed = power_transformer.transform(X_val)
X_test_power_transformed = power_transformer.transform(X_test)

# Apply Min-Max Scaling on the power-transformed features
X_train_scaled = min_max_scaler.fit_transform(X_train_power_transformed)
X_val_scaled = min_max_scaler.transform(X_val_power_transformed)
X_test_scaled = min_max_scaler.transform(X_test_power_transformed)
```

- All 3 train, test and validation sets are transformed and scaled here.
  - We can now train our model on the train set, tune it in the validation set and have a blind text on the test set.

We now train our linear regression model with the scaled data to see how the model performs.

- **Observations**:

The validation set has a MSE value of 17.1696864058348 whilst the test set has a MSE of 20.925099247848365. This indicates that the model generalizing reasonably well as there is not a difference in MSE values between the validation set and the test set. However, the model does perform slightly worse on the test set, which might be the result of the test set not having similar data patterns to the validation set.
The validation set has very similar R-squared value as validation. The difference is not very large thus indicating our model is explaining variance very well.

```
model_transformed = LinearRegression()

# Train the model on the scaled features
model_transformed.fit(X_train_scaled, y_train)

# First, make predictions on the validation set
val_predictions = model_transformed.predict(X_val_scaled)

# Evaluate the model performance on the validation set
val_mse = mean_squared_error(y_val, val_predictions)
val_r2 = r2_score(y_val, val_predictions)

print(f'Validation Mean Squared Error: {val_mse}')
print(f'Validation R-squared: {val_r2}')

if val_mse < 25 and val_r2 > 0.7:
    print("\nCriteria met. Proceeding to evaluate on the test set.\n")
    # Make predictions on the test set
    test_predictions = model_transformed.predict(X_test_scaled)

    # Evaluate the model performance on the test set
    test_mse = mean_squared_error(y_test, test_predictions)
    test_r2 = r2_score(y_test, test_predictions)

    print(f'Test Mean Squared Error: {test_mse}')
    print(f'Test R-squared: {test_r2}')
else:
    print("Model performance on validation set did not meet criteria. Consider model adjustments.")
```

```
Validation Mean Squared Error: 17.14317656601292
Validation R-squared: 0.7912860762466734

Criteria met. Proceeding to evaluate on the test set.

Test Mean Squared Error: 22.85917599088489
Test R-squared: 0.7755870744510136
```

Regularization:

We will now apply regularization using the Ridge regression to address potential overfitting and improve the model's generalization ability.

- Importance of regularization:
  - Ensures the model works effectively with blind data, as well as the data it is training.
  - From our heatmap in our EDA section, we have observed that a few features have high correlation. Regularization can distribute the coefficient weights more equally, which helps it cope with these correlated features.
  - From the boxplots in our EDA section, we have observed that features in the dataset contain outliers. Regularization helps to make the model less sensitive to this noise thus aiding the model to create more accurate predictions.
  - Below are the code and the results for regularization:

- **Observations**:
  - We have manually selected alpha of 0.1 as the base alpha value.
  - We can observe that the MSE value has slightly increased to 22.946263295314633 and the R-squared value has slightly decreased to 0.7747321216402453, and even thought the difference is very minor, it relays the fact that our model is not performing at an optimum level.
  - From this observation we can determine our next step, which is to find the optimum alpha for our Ridge Regression model.

```python
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Base alpha value selected at 0.1
alpha = 0.1

# Initializing and training the Ridge model
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(X_train_scaled, y_train)

# Making predictions on the validation set
ridge_val_predictions = ridge_model.predict(X_val_scaled)

# Evaluating the model on the validation set
ridge_val_mse = mean_squared_error(y_val, ridge_val_predictions)
ridge_val_r2 = r2_score(y_val, ridge_val_predictions)

print(f"Validation Ridge MSE: {ridge_val_mse}")
print(f"Validation Ridge R-squared: {ridge_val_r2}")

# Check to see if the validation performance is satisfactory
if ridge_val_mse < 25 and ridge_val_r2 > 0.7:

    print("\nCriteria met. Proceeding to evaluate on the test set.\n")

    # Making predictions on the test set
    ridge_test_predictions = ridge_model.predict(X_test_scaled)

    # Evaluating the model on the test set
    ridge_test_mse = mean_squared_error(y_test, ridge_test_predictions)
    ridge_test_r2 = r2_score(y_test, ridge_test_predictions)

    print(f"Test Ridge MSE: {ridge_test_mse}")
    print(f"Test Ridge R-squared: {ridge_test_r2}")

else:
    print("Model performance on validation set did not meet criteria. Consider model adjustments.")
```

```
Validation Ridge MSE: 17.563423534515586
Validation Ridge R-squared: 0.7861696736124308

Criteria met. Proceeding to evaluate on the test set.

Test Ridge MSE: 22.946263295314633
Test Ridge R-squared: 0.7747321216402453
```

Hyperparameter Tuning:

For our hyperparameter tuning, the primary focus is on optimizing the alpha parameter for Ridge regression. I have chosen to use GridSearchCV technique to do this.

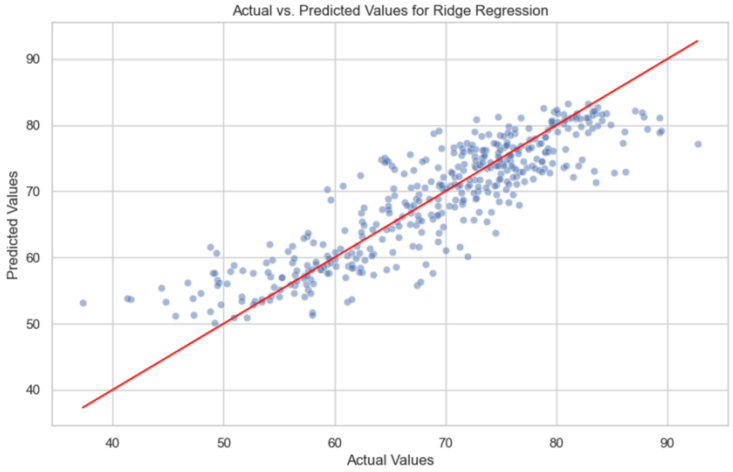
- The balance between keeping the model simple to prevent overfitting (high alpha value) and fitting the training data effectively (low alpha value) is controlled by adjusting the alpha value.
- By technically going through several alpha values and cross-validating them, the GridSearchCV approach finds the optimal value that minimises the MSE value.
- Below is the code of how I implemented this and what results I got:

```python
# Generate alpha values for Ridge
alpha_values = np.logspace(-5, 1, num=25)

# Setup the parameter grid for Ridge
param_grid_ridge = {'alpha': alpha_values}

# Initialize GridSearchCV for Ridge regression
grid_search_ridge = GridSearchCV(Ridge(random_state=3890097), param_grid_ridge, cv=5, scoring='neg_mean_squared_erro

# Fit GridSearchCV to the scaled training data
grid_search_ridge.fit(X_train_scaled, y_train)

# Evaluate the best Ridge model on the validation set instead of the test set
best_model_ridge = grid_search_ridge.best_estimator_
val_predictions_ridge = best_model_ridge.predict(X_val_scaled)
val_mse_ridge = mean_squared_error(y_val, val_predictions_ridge)
val_r2_ridge = r2_score(y_val, val_predictions_ridge)

print("Validation MSE for Best Ridge Model:", val_mse_ridge)
print("Validation R-squared for Best Ridge Model:", val_r2_ridge)

# Fit GridSearchCV to the scaled training data
grid_search_ridge.fit(X_train_scaled, y_train)

# Best alpha value found
best_alpha_ridge = grid_search_ridge.best_params_['alpha']
print("Best alpha for Ridge:", best_alpha_ridge)

# Best mean cross-validated score (negative MSE)
best_score_ridge = grid_search_ridge.best_score_
print("Best Mean Squared Error for Ridge:", -best_score_ridge)

# Evaluate the best Ridge model on the test set
best_model_ridge = grid_search_ridge.best_estimator_
predictions_ridge = best_model_ridge.predict(X_test_scaled)
mse_test_ridge = mean_squared_error(y_test, predictions_ridge)
r2_test_ridge = r2_score(y_test, predictions_ridge)
```

```python
print("Test MSE for Best Ridge Model:", mse_test_ridge)
print("Test R-squared for Best Ridge Model:", r2_test_ridge)
```

```
Validation MSE for Best Ridge Model: 17.1696864058348
Validation R-squared for Best Ridge Model: 0.7909633255204009
Best alpha for Ridge: 1e-05
Best Mean Squared Error for Ridge: 20.925099247848365
Test MSE for Best Ridge Model: 22.836952288445968
Test R-squared for Best Ridge Model: 0.7758052488105285
```

- **Observations**:
  - We have found our best alpha to be 0.00001 which is saved as best_alpha_ridge as when using this alpha on the test, we got a MSE value of 22.836952288445968 which is lower than what we got with alpha value of 0.1. The R-squared value also increased to 0.7758052488105285.
  - The alpha search range covers many orders of magnitude, ranging from 10^-5 to 10^1, offering a wide range to find a balance between prediction accuracy and model complexity.
  - My model performs well on both the validation and test sets, with a consistent level of predictive accuracy as indicated by the MSE and R-squared metrics.
  - Since the optimum alpha is very low (close to 0) it indicates that my model does not require much regularization to make accurate predictions.
  - Minimal difference between test and validation performance metrics indicates that effective generalization. Hence, the model should perform effectively on other blind datasets in the same domain.
- We can also visually represent the predicted values vs the actual values using a scatter plot graph as seen below.



  - As we can observe, the plots are closer to the y = 0 line which tells us that this is a decent fit for the model. We can also observe that the plots are not distributed very wide meaning the model's errors are within a reasonable range.

VALIDATION:

To ensure that my model works well with blind datasets, I implemented K-fold validation for the following reasons:

- In k-fold cross-validation, every data point is put in a test set in rotation exactly once and in the training set k-1 times. This is useful when we have limited data.
- In K-fold validation, the data in independent of which dataset split it will be a part of therefore it helps reduce bias in the model.
- Below is the code and the results of the K-fold validation that I conducted.

```
kf = KFold(n_splits=5, shuffle=True, random_state=3890097)
mse_scores = []  # To store the MSE scores for each fold
rmse_scores = []  # To store the RMSE scores for each fold
r2_scores = []  # To store the R-squared scores for each fold

# Perform K-Fold Cross-Validation
for train_index, test_index in kf.split(X_train_scaled):
    # Split data into training/testing sets for the current fold
    X_train_k, X_test_k = X_train_scaled[train_index], X_train_scaled[test_index]
    y_train_k, y_test_k = y_train.iloc[train_index], y_train.iloc[test_index]

    # Initialize and train the Ridge regression model on the training set
    model = Ridge(alpha=0.1)
    model.fit(X_train_k, y_train_k)

    # Predict on the testing set for the current fold
    predictions_k = model.predict(X_test_k)

    # Calculate and store the MSE for the current fold
    mse_k = mean_squared_error(y_test_k, predictions_k)
    mse_scores.append(mse_k)

    # Calculate RMSE for the current fold and store it
    rmse_scores.append(np.sqrt(mse_k))

    # Calculate and store the R-squared value for the current fold
    r2_k = r2_score(y_test_k, predictions_k)
    r2_scores.append(r2_k)

# Calculate the average MSE, RMSE, and R-squared across all folds
average_mse = np.mean(mse_scores)
average_rmse = np.mean(rmse_scores)
average_r2 = np.mean(r2_scores)

# Calculate the standard deviation of MSE, RMSE, and R-squared
std_mse = np.std(mse_scores)
std_rmse = np.std(rmse_scores)
std_r2 = np.std(r2_scores)
```

- Observations:
  - We observed the average MSE value of 21.240617842809538 and the average R-squared value of 0.757448486762965 thus able to make a close prediction to the actual value.
  - The model explained about 75.7% of the variance in the target variable.
  - The low standard deviations across MSE, RMSE, and R-squared metrics indicate that the model is consistently performing well across different data splits.

```
# Calculate the standard deviation of MSE, RMSE, and R-squared
std_mse = np.std(mse_scores)
std_rmse = np.std(rmse_scores)
std_r2 = np.std(r2_scores)

print(f"Average MSE: {average_mse}, Std. Dev. MSE: {std_mse}")
print(f"Average RMSE: {average_rmse}, Std. Dev. RMSE: {std_rmse}")
print(f"Average R-squared: {average_r2}, Std. Dev. R-squared: {std_r2}")
```

```
Average MSE: 21.240617842809538, Std. Dev. MSE: 0.5045307673493767
Average RMSE: 4.608430238137583, Std. Dev. RMSE: 0.05466793419110618
Average R-squared: 0.7574484867629651, Std. Dev. R-squared: 0.019001507082438453
```

Predictions:

With my model trained, I have used the test.csv data set to predict the life span of a human based on several attributes. The following is my code for preprocessing the new, blind, dataset which when run, creates a new csv file, and populates it with the life expectancies with respect to individual IDs.

```
new_data = pd.read_csv('./dataset/test.csv', delimiter=',')

ids = new_data['ID']

# Drop 'TARGET_LifeExpectancy' if it exists and is not needed for prediction
new_data_prepared = new_data.drop(['TARGET_LifeExpectancy'], axis=1, errors='ignore')

# Apply Power Transformation and Min-Max Scaling to the new data
new_data_power_transformed = power_transformer.transform(new_data_prepared.drop('ID', axis=1))
new_data_scaled = min_max_scaler.transform(new_data_power_transformed)

# Make predictions with the preprocessed new data using the already trained Ridge model
predictions = ridge_model.predict(new_data_scaled)

# Create a DataFrame for outputting predictions, including the 'ID' column
output_df = pd.DataFrame({'ID': ids, 'Predicted_LifeExpectancy': predictions})

# Save the predictions to a new CSV file
output_df.to_csv('predicted_life_expectancy.csv', index=False)
```

- To keep in mind:
  - ID column is stored at the beginning of and is used at the end to make the ID list.
  - Feature scaling is applied to the dataset.

Evaluation Method:

In evaluating the performance of our Ridge regression model for predicting life expectancy, we utilized three main metrics and those are Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared value. When the objective is to minimize error, MSE and RMSE offer accurate estimates of the average prediction error size.

MSE squares the differences between our predicted and actual values, thus penalizing larger errors more severely than smaller ones. This squaring is crucial as it ensures that our model's accuracy is not distorted by error directionality, providing a clear measure of prediction accuracy.

RMSE, is derived by taking the square root of MSE, and is particularly valuable due to its compatibility with the target variable's scale. This compatibility allows us to interpret the error in the same units as the target variable.

However, while MSE and RMSE give us a measure of error size, they do not offer insight into how much of the variance in the target our model can account for. This is where R-squared comes into play. It's a measure of model fit that tells us how well our independent variables capture the variability in the target variable. An R-squared value closer to 1 indicates a model that can explain a higher proportion of variance, underscoring the model's predictive power and its capacity to capture the underlying data patterns.

Since predicting life expectancy is a continuous outcome where accuracy, interpretability, and the ability to explain variance are immense, these metrics ensure we maintain a focus on minimizing error and maximizing understanding. Thus, they enable us to evaluate our model's performance, ensuring we are not just creating models that predict well but models that highlight the relationships within our data, driving forward our understanding of the factors influencing life expectancy."

Ultimate Judgement:

After a careful examination and strenuous validation process, our Ridge regression model performs brilliantly in predicting life expectancy based on many features. The model demonstrates accuracy and dependability across unknown datasets with a Validation MSE of 17.17 and an R-squared of 0.791, moving to a Test MSE of 22.84 and an R-squared of 0.776. The exploratory data analysis's observation of multicollinearity led us to the selection of Ridge regression, which has been proved by the model's robustness and generalizability. Furthermore, the ideal alpha value found during hyperparameter tuning emphasises how little regularisation the model needs, which is consistent with the linear correlations found during EDA. The model satisfies the basic premises of linear regression analysis, but it highlights areas that need additional investigation, including lowering prediction error to a greater extent and investigating the predictive capacity of other models.

REFERENCES:

1. Week 1- 4 tutorials and labs
2. Gupta, M. (2018). ML | Linear Regression - GeeksforGeeks. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/ml-linear-regression/.

3. *ML | Linear Regression vs Logistic Regression* 2020, GeeksforGeeks.

4. Gupta, M 2018, *ML | Linear Regression - GeeksforGeeks*, GeeksforGeeks.

5. *What is Exploratory Data Analysis ?* 2021, GeeksforGeeks.
6. https://www.geeksforgeeks.org/python-data-analysis-using-pandas/