ML & DL : Experiment - 2

Aim:

Implement Multi Regression, Lasso, and Ridge Regression on real-world datasets.

Dataset Description:

The Life Expectancy (WHO) Dataset is a comprehensive multivariate dataset aggregated by the World Health Organization (WHO) and the United Nations. It is designed for intermediate-to-advanced regression analysis and health informatics tasks. Unlike synthetic or "toy" datasets, this dataset represents real-world complexity, containing noisy data, missing values, and high multicollinearity, making it a standard benchmark for testing the robustness of regression algorithms (like Ridge and Lasso) and data preprocessing techniques. The dataset covers a 16-year period (from 2000 to 2015) across 193 countries, aiming to identify the key immunization, mortality, economic, and social factors that correlate with human longevity. The file type used for this dataset is a CSV (Comma Separated Values) file. The dataset consists of 22 columns, some of which are:

Column Name	Data Type	Description
Country	Categorical (String)	Name of the country where the data was recorded
Year	Numerical (Integer)	Year of observation
Status	Categorical	Development status of the country (Developing / Developed)
Life expectancy	Numerical (Float)	Average life expectancy at birth (Target variable)

Dataset Source: https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who

---

Multiple Regression

Theory:

Multiple Linear Regression is a statistical technique used to predict the outcome of a continuous dependent variable (target) based on two or more independent variables (features). It is an extension of Simple Linear Regression. While Simple Linear Regression models the relationship using a straight line, Multiple Linear Regression models the relationship using a multidimensional plane (in 3D) or a hyperplane (in higher dimensions). It allows us to understand how multiple factors simultaneously influence a single outcome. For example, predicting "Life Expectancy" (y) requires considering not just one factor, but many simultaneously, such as GDP (x_1), Schooling (x_2), and Immunization rates (x_3). Mathematical FormulationIn Simple Linear Regression, the equation is:

$$y = \beta_0 + \beta_1 x$$

In Multiple Linear Regression, we expand this to include n distinct features. The formula is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Where:

- y: The dependent variable (Target).
- x_1, x_2,..., x_n: The independent variables (Features).
- β_0: The intercept (the expected value of y when all x are O).
- β_1,..., β_n: The regression coefficients. Each coefficient β_i represents the change in the target y for a one-unit change in x_i, assuming all other variables remain constant.
- ε: The error term (residual), representing the variation in y not explained by the model.

For Multiple Regression to be valid, four key assumptions must be met:

- Linearity: The relationship between the independent and dependent variables is linear.
- No Multicollinearity: The independent variables should not be highly correlated with each other. If variables are too similar (e.g., "Age in Years" and "Age in Months"), the model cannot distinguish their individual effects.
- Homoscedasticity: The variance of the error terms is constant across all values of the independent variables.
- Normality of Errors: The residuals (errors) of the model should follow a normal distribution.

Limitations:

1. **Multicollinearity:** Multiple Linear Regression assumes that all independent variables (features) are independent of each other. Multicollinearity occurs when two or more predictor variables are highly correlated (e.g., in the WHO dataset, "GDP" and "Percentage Expenditure" are likely correlated).

   Condition of Failure: When features are highly correlated, the model struggles to determine which variable is actually responsible for the changes in the target. It makes the regression coefficients (β) unstable and unreliable. A small change in the data can lead to wild swings in the coefficients, making the model hard to interpret.
2. **The Curse of Dimensionality (Overfitting):** As you add more features to a Multiple Regression model, the model becomes more complex. While having more data is generally good, having too many features relative to the number of data points is problematic.

   Condition of Failure: If the number of features (p) approaches or exceeds the number of data points (n), the model will "memorize" the noise in the training data rather than learning the actual pattern. This leads to Overfitting, where the model performs perfectly on training data but fails badly on new, unseen data (high variance).
3. **Assumption of Feature Independence (Interaction Effects):** Multiple Regression assumes that the effect of one variable on the target is constant, regardless of the values of other variables. It treats features as additive components: $y = \beta_1 x_1 + \beta_2 x_2$.

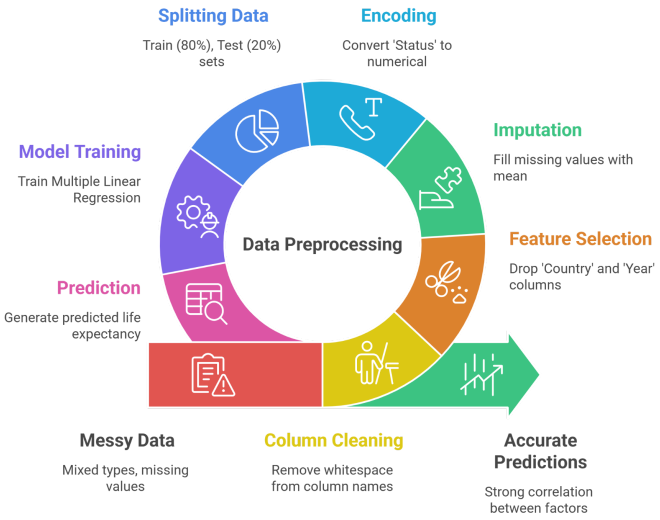
   Condition of Failure: In the real world, variables often interact. For example, in the Life Expectancy dataset, the impact of "Education" (x_1) on life expectancy might be stronger in countries with high "GDP" (x_2) and weaker in countries with low GDP. A standard additive model misses this "synergy" or interaction unless interaction terms are manually created.

Workflow:

1. **Data Collection:** The Life Expectancy dataset from WHO is loaded into a Pandas DataFrame. It contains numerical and categorical health, economic, and social indicators for 193 countries.
2. **Data Preprocessing:** Column names are cleaned to remove extra spaces. Non-predictive identifiers (Country and Year) are removed. Missing values are imputed using column means to

retain all records. The categorical Status feature (Developed/Developing) is converted into numerical form using one-hot encoding.

3. **Train–Test Split:** The dataset is split into 80% training and 20% testing data to evaluate the model on unseen samples and avoid overfitting.
4. **Model Training and Prediction:** A Multiple Linear Regression model is trained on the training set, learning optimal coefficients for all features to minimize squared error. The trained model is then used to predict life expectancy on the test set.
5. **Model Evaluation:** Performance is evaluated using RMSE to measure average prediction error (in years) and R^2 score to assess the proportion of variance explained. Actual and predicted values are compared to analyze prediction quality.



6. **Conclusion:** Multiple Linear Regression effectively captures the relationship between life expectancy and key health and economic factors. An R^2 score of 0.82 indicates strong predictive power, while remaining errors suggest that regularized models such as Ridge or Lasso could further improve performance.

Performance Analysis:

The performance of the implemented Multiple Linear Regression model on the Life Expectancy (WHO) dataset is evaluated using the Root Mean Squared Error (RMSE) and the R^2 score.

1. **Root Mean Squared Error (RMSE):**
   The obtained RMSE value is 3.90 years. RMSE represents the standard deviation of the residuals (prediction errors). In simpler terms, it measures the average distance between the predicted life expectancy values and the actual values. An RMSE of 3.90 indicates that, on average, the model's predictions are off by approximately 3.9 years. Considering that life expectancy typically ranges from 40 to 90 years, an error of roughly 4 years is acceptable for a general demographic model, though it suggests there is still room for improvement.
2. **R^2 Score (Coefficient of Determination):**
   The value for R^2 is 0.82. This means that approximately 82% of the variation in Life Expectancy can be explained by the independent variables included in the model (such as GDP, Schooling, Immunization, etc.). This indicates a strong predictive performance, as the model successfully captures the majority of the factors influencing longevity. The remaining 18% of the variation is likely due to unobserved factors not present in the dataset (such as genetics, war, political stability, or specific local healthcare quality) or non-linear relationships that this linear model cannot capture. Hence, the model demonstrates a very good fit for the real-world data.

Hyperparameter Tuning:

Standard Multiple Linear Regression (OLS - Ordinary Least Squares) has zero hyperparameters to tune. In algorithms like Random Forest or KNN, you have settings (n_estimators, k-neighbors) that change how the model learns. In Standard Linear Regression, the math finds the "best fit" line using a closed-form mathematical equation (Normal Equation) or gradient descent to minimize error. There is only one

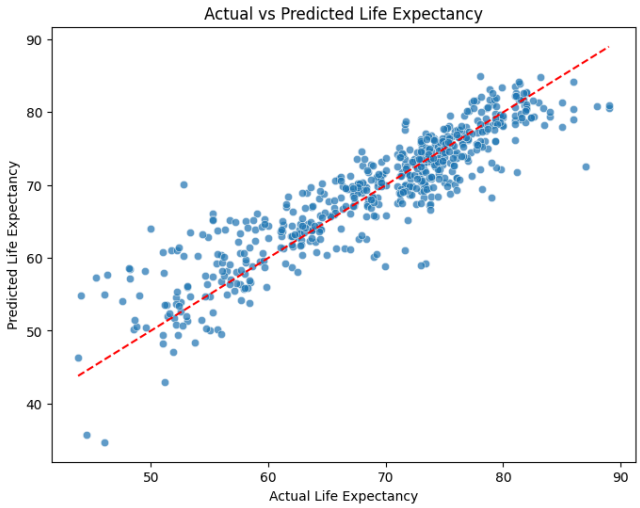
optimal solution for the provided data. You cannot "tweak" it to find a better line. If you want to tune it, you technically stop doing "Standard Linear Regression" and start doing Ridge or Lasso Regression.

Code & Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('Life Expectancy Data.csv')
df.columns = df.columns.str.strip()
df_clean = df.drop(['Country', 'Year'], axis=1)
df_clean.fillna(df_clean.mean(numeric_only=True), inplace=True)
df_final = pd.get_dummies(df_clean, columns=['Status'],
drop_first=True)
print(f"Data Shape: {df_final.shape}")
print(df_final.head())
X = df_final.drop('Life expectancy', axis=1)
y = df_final['Life expectancy']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f} years")
print(f"R² Score (Accuracy): {r2:.2f}")
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(comparison.head())
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         color='red', linestyle='--')
plt.xlabel('Actual Life Expectancy')
plt.ylabel('Predicted Life Expectancy')
plt.title('Actual vs Predicted Life Expectancy')
plt.show()
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals, alpha=0.7)
```

```
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicted Life Expectancy')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')
plt.show()
```

```
Root Mean Squared Error (RMSE): 3.90 years
R² Score (Accuracy): 0.82
      Actual  Predicted
2546    73.7  67.185528
650     75.9  77.442566
1740    74.2  75.498768
177     76.8  77.945713
1377    51.9  47.060969
```



Lasso Regression

Theory:

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a type of Linear Regression that uses a technique called Regularization to improve the model. It is specifically designed to prevent overfitting and perform Feature Selection. In standard Multiple Regression, the model tries to minimize the error without caring about how large the coefficients become. This can lead to overfitting, where the model relies too heavily on noisy or irrelevant variables. Lasso solves this by adding a "penalty" to the cost function equal to the absolute value of the coefficients. This penalty forces the coefficients to shrink towards zero. Crucially, Lasso has the unique ability to shrink coefficients exactly to zero. This means it can completely remove useless variables from the model, effectively performing automatic feature selection. The goal of Lasso is to minimize the Residual Sum of Squares (RSS) plus a penalty term.

$$Cost = \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{Error (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p} |\beta_j|}_{\text{L1 Penalty}}$$

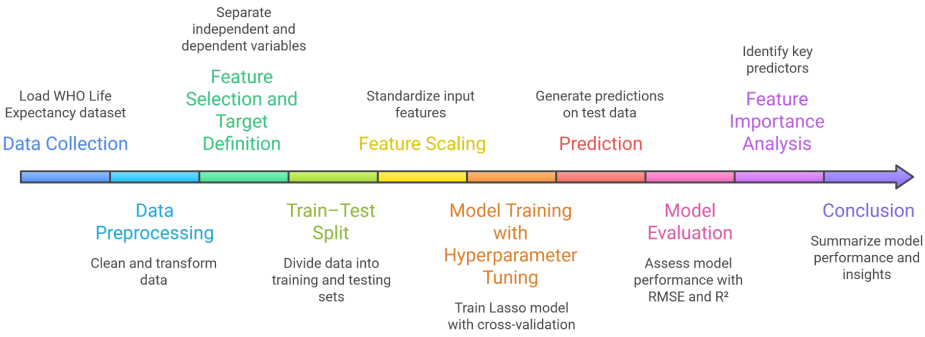
Limitations:

1. **Handling of Correlated Variables (Arbitrary Selection):** Lasso struggles when features are highly correlated (Multicollinearity). If two variables (e.g., "GDP" and "Healthcare Spend") are strongly correlated, Lasso tends to pick one of them arbitrarily and shrink the other to zero.
   Condition of Failure: It does not treat them as a group. This can lead to loss of interpretability because the variable it kept might not be the most scientifically relevant one, just the one that was mathematically slightly better in that specific dataset.
2. **Number of Predictors vs. Observations (p>n):** If the number of features (p) is greater than the number of data points (n), Lasso has a hard mathematical limit.
   Condition of Failure: Lasso can select at most n variables before the optimization saturates. It cannot identify more important features than there are data samples, which is a significant drawback in high-dimensional genomics or text data.

Workflow:

1. **Data Collection:** The WHO Life Expectancy dataset is loaded into a Pandas DataFrame containing health, economic, and social indicators.
2. **Data Preprocessing:** Column names are cleaned, non-informative columns (Country and Year) are removed, missing values are imputed using column means, and the categorical Status feature is encoded using one-hot encoding.
3. **Train–Test Split and Scaling:** The data is split into 80% training and 20% testing sets. All features are standardized using StandardScaler, which is required for Lasso regression.
4. **Model Training and Prediction:** Lasso regression with cross-validation (LassoCV) is trained to automatically select the optimal regularization hyperparameter (alpha). The trained model is used to predict life expectancy on the test data.

5. **Model Evaluation and Feature Selection:** Performance is evaluated using RMSE and R^2 score. Lasso coefficients are analyzed to identify important features, with irrelevant variables eliminated by shrinking their coefficients to zero.
6. **Conclusion:** Lasso regression achieves strong predictive performance (R^2 = 0.82, RMSE ≈ 3.9 years) while simultaneously performing feature selection, making it effective for modeling life expectancy.



Performance Analysis:

The performance of the implemented Lasso Regression model is evaluated using Root Mean Squared Error (RMSE) and the R^2 score, which together assess prediction accuracy and explanatory power. These metrics quantify how well the model predicts life expectancy based on health and socio-economic indicators.

1. **RMSE Analysis:** The model achieved an RMSE of approximately 3.9 years, indicating that the predicted life expectancy values deviate from the actual values by about 3.9 years on average. Given the complexity and real-world variability of public health data, this error margin is considered low and acceptable.
2. **R^2 Score Analysis:** The R^2 score of 0.82 indicates that the model explains 82% of the variance in life expectancy. This demonstrates a strong linear relationship between the selected predictors and the target variable, confirming effective learning from the data.

Hyperparameter Tuning:

Hyperparameter tuning was performed to optimize the Lasso Regression model and improve its generalization performance. Instead of relying on default parameters, LassoCV was used to automatically select the optimal regularization strength through k-fold cross-validation. This approach reduces the risk of overfitting and ensures a balanced trade-off between model complexity and prediction accuracy.

Hyperparameter Tuned: Alpha (α) – Regularization Strength. Alpha controls the magnitude of the L1 penalty applied to the regression coefficients.

1. Smaller α values: Weaker regularization, retaining more features.
2. Larger α values: Stronger regularization, shrinking more coefficients to zero and performing feature selection.

A logarithmic range of 50 alpha values between 0.0001 and 10 was evaluated using 5-fold cross-validation to identify the optimal value. The optimal Hyperparameter Value is:

Best Alpha (α): 0.00010

This value indicates that the dataset benefits from mild regularization, suggesting that most features contribute meaningfully to life expectancy prediction while still controlling overfitting.

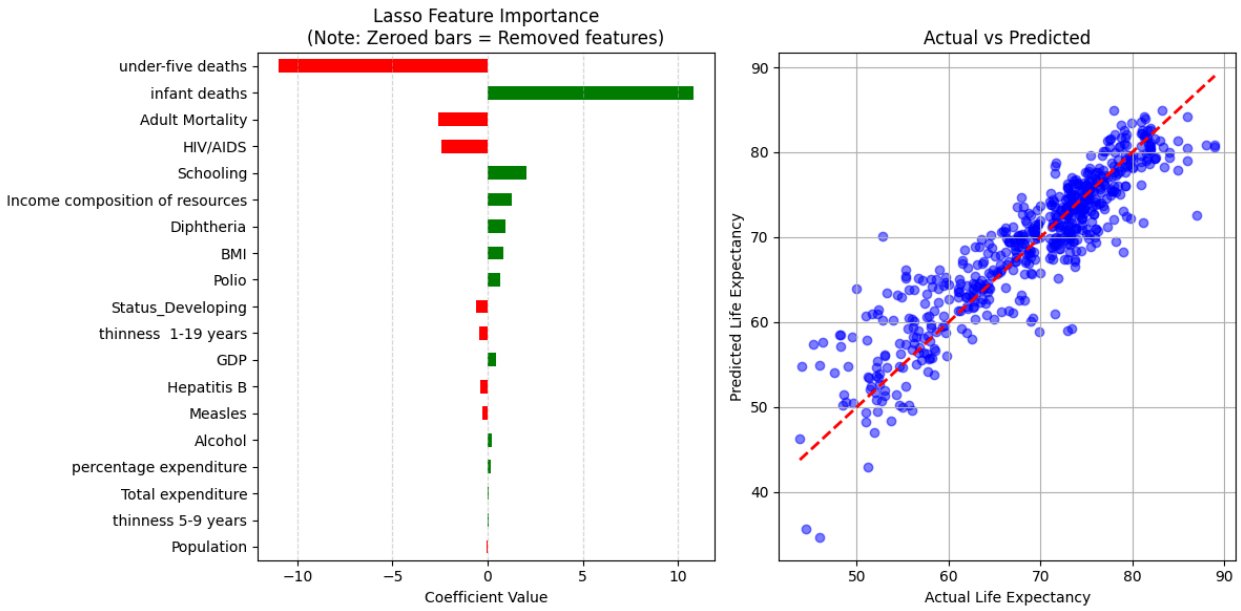
.

Code & Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv('Life Expectancy Data.csv')

df.columns = df.columns.str.strip() # Remove whitespace
df = df.drop(['Country', 'Year'], axis=1) # Drop identifiers
df.fillna(df.mean(numeric_only=True), inplace=True) # Handle Missing
Values
df = pd.get_dummies(df, columns=['Status'], drop_first=True) # One-Hot
Encoding

X = df.drop('Life expectancy', axis=1)
y = df['Life expectancy']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lasso_cv = LassoCV(alphas=np.logspace(-4, 1, 50), cv=5,
random_state=42)

lasso_cv.fit(X_train_scaled, y_train)

print(f"Optimal Alpha (Hyperparameter) found: {lasso_cv.alpha_:.5f}")
y_pred = lasso_cv.predict(X_test_scaled)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Lasso RMSE: {rmse:.2f} years")
print(f"Lasso R² Score: {r2:.2f}")

plt.figure(figsize=(12, 6))
```

```
coefs = pd.Series(lasso_cv.coef_, index=X.columns)
coefs_sorted =
coefs.reindex(coefs.abs().sort_values(ascending=True).index)

plt.subplot(1, 2, 1)
coefs_sorted.plot(kind='barh', color=(coefs_sorted > 0).map({True:
'green', False: 'red'}))
plt.title(f"Lasso Feature Importance\n(Note: Zeroed bars = Removed
features)")
plt.xlabel("Coefficient Value")
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred, alpha=0.5, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Life Expectancy")
plt.ylabel("Predicted Life Expectancy")
plt.title("Actual vs Predicted")
plt.grid(True)

plt.tight_layout()
plt.savefig('lasso_results.png')
plt.show()

print("\nFeatures eliminated by Lasso (Coeff ~ 0):")
print(coefs[coefs == 0].index.tolist())
```

```
Optimal Alpha (Hyperparameter) found: 0.00010
Lasso RMSE: 3.90 years
Lasso R² Score: 0.82
```



Ridge Regression

Theory:

Ridge Regression (L2 Regularization) is a technique used to analyze multiple regression data that suffer from multicollinearity (independent variables are highly correlated). When multicollinearity exists, standard regression estimates are unbiased but have huge variances, meaning they are far from the true value. Ridge Regression adds a different kind of "penalty" to the model than Lasso. While Lasso penalizes the absolute value of coefficients, Ridge penalizes the square of the magnitude of coefficients. Ridge shrinks the coefficients toward zero, but it never makes them exactly zero. This means Ridge Regression will always include all the features in the final model, just with smaller weights. It is a method for Coefficient Shrinkage, not Feature Selection. The goal of Ridge is to minimize the Residual Sum of Squares (RSS) plus the squared penalty.

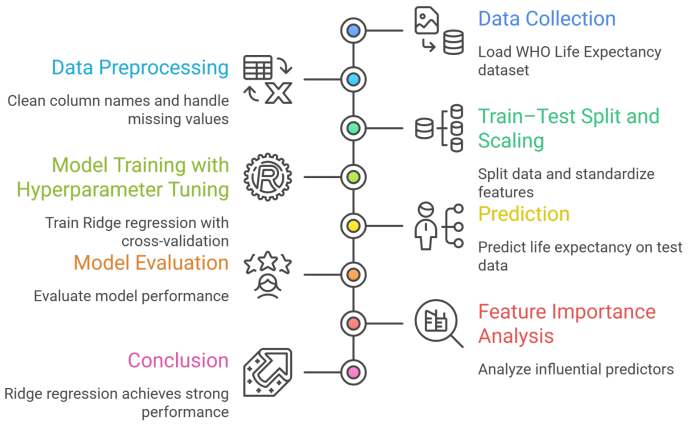
$$Cost = \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{Error (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{\text{L2 Penalty}}$$

Limitations:

1. **Lack of Feature Selection (Model Interpretability):** The biggest limitation of Ridge compared to Lasso is that it cannot set coefficients to exactly zero.
   Condition of Failure: If you have a dataset with 1,000 features, but only 5 are actually important, Ridge will still give you a model with 1,000 features (995 of them will just be very small). This makes the model dense and very hard to interpret or explain to a stakeholder.
2. **Bias-Variance Tradeoff:** Like all regularization methods, Ridge deliberately introduces bias into the model to reduce variance.
   Condition of Failure: If the λ value is set too high, the model becomes too rigid (underfitting). It will fail to capture important signals in the data because it is too busy punishing the coefficients.

Workflow:

1. **Data Collection:** The WHO Life Expectancy dataset is loaded into a Pandas DataFrame containing health, economic, and social indicators.
2. **Data Preprocessing:** Column names are cleaned, non-predictive columns (Country and Year) are removed, missing values are imputed using column means, and the categorical Status feature is encoded using one-hot encoding.
3. **Train–Test Split and Scaling:** The data is split into 80% training and 20% testing sets. All features are standardized using StandardScaler, which is required for Ridge regression.



4. **Model Training and Prediction:** Ridge regression with cross-validation (RidgeCV) is trained to automatically select the optimal regularization hyperparameter (alpha). The trained model is then used to predict life expectancy on the test data.
5. **Model Evaluation and Interpretation:** Performance is evaluated using RMSE and R^2 score. Ridge coefficients are analyzed to identify important predictors, noting that coefficients are shrunk but not reduced to zero.
6. **Conclusion:** Ridge regression achieves strong predictive performance (R^2 = 0.82, RMSE ≈ 3.9 years) while effectively handling multicollinearity among features..

Performance Analysis:

The performance of the implemented Ridge Regression model is evaluated using Root Mean Squared Error (RMSE) and the R^2 score, which measure prediction accuracy and explanatory power.

1. **RMSE Analysis:** The model achieved an RMSE of approximately 3.9 years, indicating that the predicted life expectancy values differ from the actual values by about 3.9 years on average. This low error suggests that the model provides accurate predictions despite the inherent variability in public health data.
2. **R^2 Score Analysis:** An R^2 score of 0.82 indicates that Ridge regression explains 82% of the variance in life expectancy. This demonstrates a strong linear relationship between the predictors and the target variable and confirms effective learning from the dataset.

Hyperparameter Tuning:

Hyperparameter tuning was performed to optimize the Ridge Regression model and improve its generalization performance. Instead of using default parameters, RidgeCV was employed to automatically select the optimal regularization strength through k-fold cross-validation. This approach ensures a balance between model complexity and prediction accuracy while reducing overfitting.

Hyperparameter Tuned: Alpha (α) – Regularization Strength:

Alpha controls the magnitude of the L2 penalty applied to regression coefficients.

- Smaller α values: Weaker regularization, allowing coefficients to remain closer to their original values.
- Larger α values: Stronger regularization, shrinking coefficients more aggressively to handle multicollinearity.

A logarithmic range of 50 alpha values between 0.01 and 100 was evaluated using 5-fold cross-validation to identify the optimal value.

Optimal Hyperparameter Value:

Best Alpha (α): 0.01

This value indicates that moderate regularization is sufficient to stabilize coefficient estimates while preserving all predictive features.

Code & Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv('Life Expectancy Data.csv')

df.columns = df.columns.str.strip() # Remove whitespace
df = df.drop(['Country', 'Year'], axis=1) # Drop identifiers
df.fillna(df.mean(numeric_only=True), inplace=True) # Handle Missing
Values
df = pd.get_dummies(df, columns=['Status'], drop_first=True) # One-Hot
Encoding

X = df.drop('Life expectancy', axis=1)
y = df['Life expectancy']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

alphas_to_test = np.logspace(-2, 2, 50) # 50 values between 0.01 and
100
ridge_cv = RidgeCV(alphas=alphas_to_test, cv=5)

ridge_cv.fit(X_train_scaled, y_train)

print(f"Optimal Alpha (Hyperparameter) found: {ridge_cv.alpha_:.5f}")
y_pred = ridge_cv.predict(X_test_scaled)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Ridge RMSE: {rmse:.2f} years")
print(f"Ridge R² Score: {r2:.2f}")
```

```
plt.figure(figsize=(12, 6))

coefs = pd.Series(ridge_cv.coef_, index=X.columns)
coefs_sorted =
coefs.reindex(coefs.abs().sort_values(ascending=True).index)

plt.subplot(1, 2, 1)
coefs_sorted.plot(kind='barh', color=(coefs_sorted > 0).map({True:
'blue', False: 'orange'}))
plt.title(f"Ridge Feature Importance\n(Note: No feature is exactly
0)")
plt.xlabel("Coefficient Value")
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred, alpha=0.5, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Life Expectancy")
plt.ylabel("Predicted Life Expectancy")
plt.title("Actual vs Predicted")
plt.grid(True)

plt.tight_layout()
plt.savefig('ridge_results.png')
plt.show()

print("\nTop 5 Most Important Features (Ridge):")
print(coefs.abs().sort_values(ascending=False).head(5))
```

```
Top 5 Most Important Features (Ridge):
under-five deaths    11.044418
infant deaths        10.852937
Adult Mortality       2.591541
HIV/AIDS              2.420340
Schooling             2.060673
dtype: float64
```

```
Optimal Alpha (Hyperparameter) found: 0.01000
Ridge RMSE: 3.90 years
Ridge R² Score: 0.82
```